KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculty of Business and Economics

Exact and heuristic reactive planning procedures for multimode resource-constrained projects

F. Deblaere, E. Demeulemeester and W. Herroelen

DEPARTMENT OF DECISION SCIENCES AND INFORMATION MANAGEMENT (KBI)

KBI 0818

Exact and heuristic reactive planning procedures for multi-mode resource-constrained projects

Filip Deblaere Erik Demeulemeester
Willy Herroelen
Research Center for Operations Management
Department of Decision Sciences and Information Management
Faculty of Business and Economics
Katholieke Universiteit Leuven (Belgium)

August 12, 2008

Abstract

The multi-mode resource-constrained project scheduling problem (MRCPSP) involves the determination of a baseline schedule of the project activities, which can be executed in multiple modes, satisfying the precedence relations and resource constraints while minimizing the project duration. During the execution of the project, the baseline schedule may become infeasible due to activity duration and resource disruptions. We propose and evaluate a number of dedicated exact reactive scheduling procedures as well as a tabu search heuristic for repairing a disrupted schedule. We report on promising computational results obtained on a set of benchmark problems.

1 Introduction

The resource-constrained project scheduling problem or RCPSP (problem $m, 1|cpm|C_{max}$ in the notation of Herroelen et al. (2000)) involves the scheduling of project activities subject to resource and precedence constraints, under the objective of minimizing the project makespan. In the multi-mode RCPSP or MRCPSP (problem $m, 1T|cpm, mu|C_{max}$), activities can be executed in

multiple modes and the availability of resource types may not only be specified per time period, but also for the entire project horizon T (in that case, we speak of *nonrenewable* resource types). The activity modes are characterized by a certain resource requirement and a duration. For instance, one can either build a wall with one worker in three days or with two workers in two days. A number of solution procedures for the deterministic MRCPSP have been proposed in the literature, both exact and heuristic. We refer the interested reader to Speranza and Vercellis (1993), Drexl and Grünewald (1993), Hartmann and Sprecher (1996), Sprecher et al. (1997), Hartmann and Drexl (1998), Sprecher and Drexl (1998), De Reyck and Herroelen (1999), Hartmann (2001), Jozefowska et al. (2001), Alcaraz et al. (2003), Heilmann (2003), Buddhakulsomsiri and Kim (2006), Lova et al. (2006, 2008), Zhang et al. (2006), Zhu et al. (2006) and Sabzehparvar and Seyed-Hosseini (2008).

In recent years there has been a growing conscience that the traditional deterministic project scheduling models are in plain conflict with a reality that is characterized by uncertainty. As correctly noted by Hildum (1995), an optimal schedule is only optimal to the extent that the reality behaves as expected during the execution of the schedule. For review papers on production scheduling under uncertainty, we refer to Davenport and Beck (2002), Vieira et al. (2003) and Aytug et al. (2005). For a review of project scheduling under uncertainty, we refer the reader to Herroelen and Leus (2004, 2005).

In project scheduling, uncertainty can take many different forms. Activity duration estimates may be off, resources may break down, work may be interrupted due to weather delay, new unanticipated activities may be identified, etc. All of these types of uncertainty may result in the infeasibility of the project baseline schedule. In general, project management wants to avoid these schedule breakages. This can be achieved by generating a baseline schedule in a *proactive* way, trying to anticipate certain types of disruptions so as to minimize their effect if they occur. If the schedule would still break down despite these proactive planning efforts, a *reactive* scheduling policy will be needed to repair the infeasible schedule.

In the field of proactive and reactive project scheduling for the single-mode RCPSP some work has already been done. The problem of coping with activity duration variability has been tackled in Van de Vonder (2006) and Van de Vonder et al. (2007, 2008). The problem of uncertainty with respect to resource availability has been addressed by Lambrechts et al. (2007, 2008). The literature on proactive/reactive scheduling policies in the multi-mode RCPSP, however, is virtually void. To the best of our knowledge, there is only the work by Zhu et al. (2005), where a branch-and-cut procedure is proposed for a general class of reactive scheduling problems. The authors

assume the presence of a *recovery window* that limits the set of feasible reactive schedules to those schedules that are "back on track" from a certain time point onward. Computational results are reported for the generation of reactive schedules in response to a duration disruption on a set of 20-activity instances.

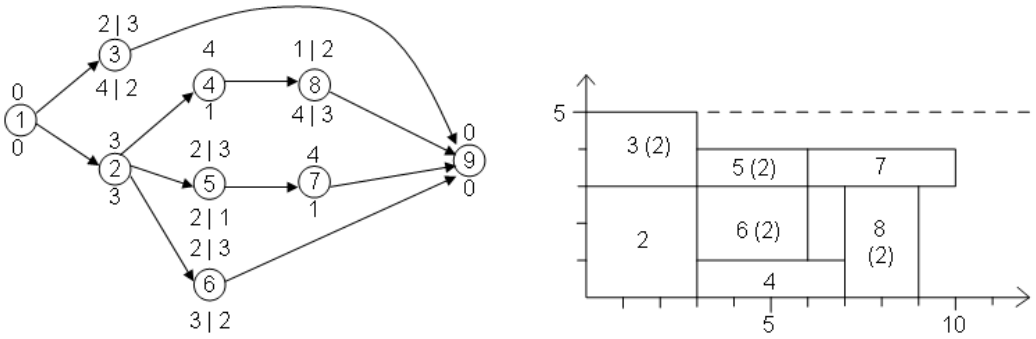
In this paper, we propose a number of solution procedures for the multi-mode reactive scheduling problem that are capable of solving 20-job as well as 30-job benchmark instances within a reasonable computation time. The procedures are able to handle both activity duration disruptions as well as resource disruptions. The structure of the paper is as follows. In Section 2, we give a description of the basic MRCPSP and the associated reactive scheduling problem. For solving this problem, a number of dedicated exact search procedures as wel as a tabu search heuristic are proposed in Section 3. In Section 4 we test the procedures on 20- and 30-activity benchmark problems. We also comment on some general characteristics of optimal reactive schedules. In the final section, we provide overall conclusions.

2 Problem description

2.1 The basic deterministic MRCPSP

The basic deterministic MRCPSP can be described as follows. We assume that the project is represented as an activity-on-the-node network $G(N, A)$ with a set of nodes $N = \{1, 2, \ldots, n\}$ representing the project activities and a set of arcs A representing the zero-lag finish-start precedence constraints between the activities. We assume that activities 1 and n denote the dummy start and the dummy end activity, respectively. We denote M_i as the set of available modes for an activity i, each mode being the combination of a certain activity duration and a certain resource requirement. The duration of an activity i executed in a mode $m_i \in M_i$ is denoted as d_{im_i}.

We assume the presence of a set $\mathcal{K}^\rho$ of *renewable* resource types and a set $\mathcal{K}^\nu$ of *nonrenewable* resource types. Contrary to renewable resource types, the availability of nonrenewable resource types is specified for the entire project horizon T. These resource types are useful for modelling e.g. a limited budget for the execution of the project. Extra activity execution costs (such as overtime or weekend work) corresponding with shorter execution modes can then be modelled through the use of a higher amount of these nonrenewable resources. *Doubly-constrained* resource types can be considered as the resource types in the intersection of $\mathcal{K}^\nu$ and $\mathcal{K}^\rho$. Hence, we do not need to consider them explicitly. The project activities $i \in N$ executed in



(a) Project network (b) Baseline schedule

Figure 1: An example MRCPSP instance

a mode $m_i \in M_i$ require an integer per-period amount $r^{\rho}_{im_ik}$ of the renewable resource type k, $k \in \mathcal{K}^{\rho}$ and an integer amount $r^{\nu}_{im_ik}$ of the nonrenewable resource type k, $k \in \mathcal{K}^{\nu}$. We denote a^{ρ}_k as the per-period availability of the renewable resource type k ($k \in \mathcal{K}^{\rho}$) and a^{ν}_k as the total availability of the nonrenewable resource type k ($k \in \mathcal{K}^{\nu}$).

The objective of the basic MRCPSP is to determine a schedule consisting of activity starting times s_i and activity execution modes m_i ($m_i \in M_i$) such that the precedence as well as the resource constraints are satisfied, while minimizing the project makespan. An example MRCPSP instance along with a (suboptimal) baseline schedule is shown in Figure 1. The example project depicted in Figure 1 (a) has one renewable resource type with a per-period availability equal to five units and no nonrenewable resource types. The possible activity durations are shown above the activity nodes and the corresponding resource requirements per time period are shown below the nodes. When multiple execution modes are possible, these are separated by a vertical bar. For instance, activity 3 can either be executed in two time units using four units of resource per period or in three time units using only two units of resource per period. Activities 1 and 9 are the dummy start and the dummy end activity, respectively. The resource profile of the baseline schedule is shown in Figure 1 (b). Whenever multiple activity execution modes are possible, the selected mode is indicated between brackets. The example project shown in Figure 1 (a) along with its baseline schedule shown in Figure 1 (b) will be used as an illustrative example throughout this paper.

2.2 Schedule disruptions and rescheduling costs

In a realistic project environment one has to cope with a considerable amount of uncertainty. If an unanticipated disruptive event occurs (e.g. a machine

i	1	2	3	4	5	6	7	8	9
w_i	0	2	1	2	2	5	1	12	15
c_{i1}	0	0	5	0	4	1	0	4	0
c_{i2}	-	-	0	-	0	0	-	0	-

Table 1: Inflexibility weights and mode switching costs.

breakdown), the baseline schedule may become infeasible. Project management must then rely on a *reactive procedure* to adapt the schedule to the new information and to restore its feasibility. In most situations it is important that the repaired schedule bears close resemblance to the original baseline schedule. Any deviation from the baseline schedule may lead to undesirable side-effects, such as having to change agreements with subcontractors, accumulating inventory costs, dealing with employee malcontent, etc. Moreover, if resources with scarce availability need to be reserved in advance, any disruption of the schedule may lead to a large delay of the activities in need of the scarce resources, which may ultimately lead to a violation of the project deadline. We can capture the magnitude of the costs incurred when changing the starting time of an activity $i \in N$ by one time unit through a nonnegative *inflexibility weight* w_i.

In a multi-mode context, in addition to delaying activities in order to repair a broken schedule, we might also consider changing the mode of certain activities. Indeed, changing an activity's mode can enable us to speed up the execution of that activity, allowing us to get the schedule back on track sooner, at the price of a certain *mode switching cost* $c_{i(m_i,m'_i)}$ reflecting the cost of switching the baseline mode m_i of activity i to the reactive mode m'_i. Since we assume that the baseline modes of all activities i are given, we can simplify the notation of the mode switching costs to $c_{im'_i}$. We assume that no mode switching cost is incurred when the reactive mode is the same as the baseline mode. In other words, we assume $c_{im_i} = 0, \forall i \in N$. This reflects our desire not to change the mode of an activity unless this results in a reduced cost incurred due to activity starting time deviations. The inflexibility weights and the mode switching costs for the example instance of Figure 1 are shown in Table 1. Note that in the baseline schedule of Figure 1 (b) all activities with two modes are executed in their second mode, hence we have $c_{i2} = 0$ for these activities. Evidently, mode switching costs are irrelevant for activities with only one execution mode.

2.2.1 Objective function

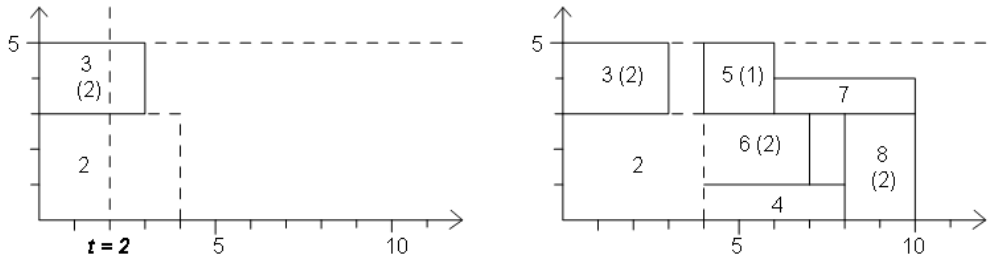
Given a baseline schedule of activity starting times $s_1, s_2, \ldots, s_n$ and execution modes $m_1, m_2, \ldots, m_n$, the objective of the reactive scheduling procedure we propose is to produce a revised schedule of reactive starting times $s'_1, \ldots, s'_n$ and execution modes $m'_1, \ldots, m'_n$ such that the *rescheduling cost* $\mathcal{C}$ given by Equation (1) is minimized.

$$\mathcal{C} = \sum_{i \in N} w_i |s'_i - s_i| + \sum_{i \in N} c_{im'_i} \tag{1}$$

Clearly, the rescheduling cost (1) is a function of the decision variables s'_i and m'_i, $i = 1, \ldots, n$. As an additional constraint on the reactive schedule, we require for each activity i that $s'_i \geq s_i$. This constraint is commonly referred to as the *railroad scheduling policy*, due to the analogy with railroad scheduling, where trains do not depart before their scheduled time of departure. Including this constraint transforms the objective function (1) into a regular performance measure, which will enable us to include some powerful dominance rules in our procedure. This extra constraint will exclude some schedules as possible solutions to the reactive scheduling problem, but it should be noted that in a disrupted schedule one will often not have the luxury of starting an activity before its baseline starting time. Moreover, in a practical context, it will often be impossible to start activities earlier than planned, for instance because the necessary materials have not yet been delivered to the site or because of inflexible agreements with subcontractors.

2.2.2 Activity duration disruptions

In this paper two types of schedule disruptions will be explicitly considered, namely activity duration disruptions and resource disruptions. In the case of an activity duration disruption we assume that during the execution of some activity i, it becomes clear that the duration of that activity will be an amount Δ_i higher than its deterministic baseline activity duration d_{im_i}. This case is illustrated in Figure 2. At time instant $t = 2$ it becomes clear that activity 2 will take four instead of three time units to complete ($\Delta_2 = 1$). This disruption is illustrated in Figure 2 (a). Figure 2 (b) presents an optimal reactive schedule, in which the starting times of activities 4, 5, 6 and 8 are delayed for one time unit. The mode of activity 5 has been switched, enabling activity 7 to start at its baseline starting time, which ensures that the baseline project makespan of 10 time units is met. The reader can verify that for this example the rescheduling cost $\mathcal{C}$ equals 25.



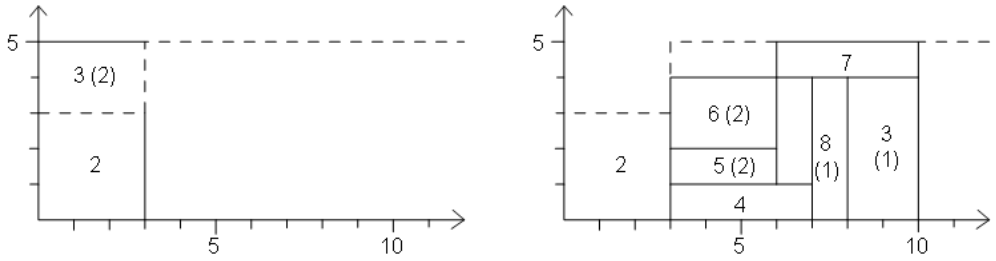
(a) Disruption of activity 2 (b) Reactive schedule

Figure 2: Activity duration disruption

2.2.3 Renewable resource disruptions

In the case of a renewable resource disruption we assume that the availability of a renewable resource type $k \in \mathcal{K}^\rho$ at time instant t suddenly decreases by an amount Δ_k^ρ. We furthermore assume that the project manager is able to make a prediction of the time period $t + \delta_t$ where the availability of that resource type will be restored to its original level. If the disruption is sufficiently severe, we may not be able to continue the execution of the project as planned in the baseline schedule. In that case we need to select a set of activities that are active at time instant t such that, when delaying the execution of these activities to a later point in time, the remaining activities that are active at time instant t can be executed within the reduced resource availability. Given such a (minimal) delaying alternative, renewable resource disruptions can be treated in a way that is very similar to activity duration disruptions. For reasons of clarity, we assume a preempt-repeat environment, where interrupted activities must be re-executed from scratch. Although this is perhaps not the most realistic setting, using this assumption will allow us to avoid issues regarding residual work content of preempted activities that would arise in a preempt-resume environment.

An example of a renewable resource disruption is given in Figure 3. Already at time instant $t = 0$ the availability of the single renewable resource type drops with an amount $\Delta_1^\rho = 2$ and is predicted to be restored at time instant $t + \delta_t = 3$, as shown in Figure 3 (a). As a consequence, the execution of activities 2 and 3 cannot be continued as planned and reactive measures need to be taken. An optimal reactive schedule is presented in Figure 3 (b). The highly flexible activity 3 ($w_3 = 1$, see Table 1) is delayed for eight time units, until time instant $t = 8$. By switching the modes of activities 3 and 8, we are again able to meet the projected makespan of 10 time units. This set of reactive measures results in a total rescheduling cost $\mathcal{C} = 17$.



(a) Resource disruption at $t = 0$ (b) Reactive schedule

Figure 3: Resource disruption

2.2.4 Nonrenewable resource disruptions

In the case of a nonrenewable resource disruption, it is assumed that the availability of a nonrenewable resource type $k \in \mathcal{K}^{\nu}$ drops with an amount Δ_k^{ν}. If this results in an infeasible schedule with respect to the nonrenewable resources, a reactive schedule must be sought such that the reactive modes allow all activities to be executed within the reduced nonrenewable resource availability, while the resulting rescheduling costs are minimized.

3 Tree-based exact search techniques for the reactive scheduling problem

3.1 Branching scheme

We assume that the baseline schedule is executed as planned, up to a certain decision time t^*, at which a disruption occurs. As mentioned in the previous section, this can either be an activity duration disruption or a resource disruption. The objective is then to construct a reactive schedule of the unfinished activities that is feasible with respect to the newly available information, and minimizes the rescheduling cost $\mathcal{C}$ as defined in Equation (1). To solve this reactive scheduling problem tree-based search techniques can be used that are similar to those devised for solving the basic MRCPSP.

As mentioned in Hartmann and Drexl (1998), three categories of branching schemes for the MRCPSP are described in the literature, namely precedence tree enumeration, branching on mode and delaying alternatives, and branching on mode and extension alternatives. In this paper, we propose a branching procedure based on mode and delaying alternatives. Before we discuss the structure of the branching scheme, we need some additional terminology and notation. Given are a baseline schedule of activity starting times

$s_1, \ldots, s_n$ and a set of baseline modes $m_1, \ldots, m_n$. In a reactive schedule the baseline starting times s_i become release dates τ_i and the railroad scheduling constraint on the reactive activity starting times can thus be formulated as $s'_i \geq \tau_i$, $\forall i \in N$.

In the search tree we will associate with each level l of the tree a partial reactive schedule PS_l, a set of pending activities $\mathcal{P}_l$, a set of eligible activities $\mathcal{E}_l$ and a set of active activities $\mathcal{A}_l$. The partial reactive schedule PS_l is characterized by a decision time t_l, a set $\mathcal{S}_l$ of the scheduled activities, a set of reactive starting times s'_i, $\forall i \in \mathcal{S}_l$ and a set of reactive modes m'_i, $\forall i \in \mathcal{S}_l$. From a partial reactive schedule PS_l we can calculate $\mathcal{P}_l$, the set of *pending* activities at time instant t_l: this is the set of unscheduled activities for which all predecessors have already finished at or before time instant t_l. The set $\mathcal{E}_l \subseteq \mathcal{P}_l$ of *eligible* activities is then defined as: $\mathcal{E}_l = \{i \in \mathcal{P}_l \,|\, t_l \geq \tau_i\}$. For the activities in $\mathcal{E}_l$, all mode alternatives will be considered. A *mode alternative* $\mathcal{M}_{lj}$ is a mapping that assigns a certain mode $\mathcal{M}_{lj}(i) \in M_i$ to every activity $i \in \mathcal{E}_l$. Given a mode alternative, starting all activities in $\mathcal{E}_l$ may lead to a renewable resource conflict. To solve this resource conflict, a number of minimal delaying alternatives will be identified. By delaying the start of the activities of such a minimal delaying alternative, we can eliminate the resource conflict at the current decision time. Formally, a *delaying alternative* $\mathcal{D}_{lj}$ is a subset of the set of active activities $\mathcal{A}_l$ such that $\sum_{i \in \mathcal{A}_l \setminus \mathcal{D}_{lj}} r^{\rho}_{im'_i k} \leq a^{\rho}_k$, $\forall k \in \mathcal{K}^{\rho}$. A delaying alternative $\mathcal{D}_{lj}$ is called *minimal* if no proper subset of $\mathcal{D}_{lj}$ is also a delaying alternative. Given this notation and terminology, we can now present the formal structure of the branching scheme. The detailed steps of the procedure are shown in Algorithm 1.

3.2 Lower bound

When a first feasible solution for the reactive scheduling problem is found, the objective value of that solution can be used as an upper bound. Nodes in the search tree can then be dominated by means of the lower bound on the rescheduling cost that is computed by means of Algorithm 3. The lower bound consists of the sum of two terms. The first term simply consists of the rescheduling costs that directly result from the scheduling decisions involving the activities in $\mathcal{S}_l$, the set of scheduled activities in the partial reactive schedule. The second term consists of unavoidable rescheduling costs due to activity starting time deviations of the unscheduled activities in PS_l. To calculate this term, we need the earliest starting times es_i of the unscheduled activities $i \in (N \setminus \mathcal{S}_l)$. These starting times are calculated ignoring the renewable resource constraints and assuming that all unscheduled activities i are executed in their shortest nonrenewable resource feasible mode m^*_i. Clearly,

Algorithm 1 Reactive scheduling procedure

set $l = 1$, $\mathcal{S}_1 = \{i \in N | s_i \leq t^*\}$ and $\mathcal{A}_1 = \{i \in \mathcal{S}_1 | s_i + d_{im_i} > t^*\}$
$\forall i \in \mathcal{S}_1$, set $s'_i = s_i$ and $m'_i = m_i$. $\forall i \in N:$ set $\tau_i = s_i$

- in the case of a **duration disruption** of activity i:
  set $d_{im'_i} = d_{im'_i} + \Delta_i$
  perform Algorithm 2 (branching on mode and delaying alternatives)
- in the case of a **renewable resource disruption**:
  identify all minimal delaying alternatives $\mathcal{D}_j$ that solve the resource conflict caused by the resource disruption
  for every $\mathcal{D}_j$:
    set $\mathcal{S}_1 = \mathcal{S}_1 \setminus \mathcal{D}_j$, $\mathcal{A}_1 = \mathcal{A}_1 \setminus \mathcal{D}_j$
    $\forall i \in \mathcal{D}_j$, discard the mode assignment made to activity i
    perform Algorithm 2 (branching on mode and delaying alternatives)
- in the case of a **nonrenewable resource disruption**:
  given the affected resource type k, set $a^\nu_k = a^\nu_k - \Delta^\nu_k$
  perform Algorithm 2 (branching on mode and delaying alternatives)

as the earliest starting times es_i are lower bounds on the actual reactive starting times s'_i obtainable given the partial reactive schedule, Algorithm 3 gives us a lower bound on the rescheduling cost $\mathcal{C}$ obtainable by branching from PS_l.

The lower bound can be calculated in Step 4 of Algorithm 2 if the current mode assignment does not result in a renewable resource conflict, or in Step 6 otherwise. Evidently, whenever a new best solution is found, the upper bound should be updated as well.

3.3 Dominance rules

3.3.1 Data reduction

A first set of dominance criteria can be achieved implicitly by pre-processing the input data. Redundant nonrenewable resource types can be identified, and inefficient and non-executable activity modes can be removed. To define these concepts, we need the notion of the minimal requirement $rmin^\nu_{ik}$ of an activity i for the nonrenewable resource type k: $rmin^\nu_{ik} = \min_{m_i \in M_i}(r^\nu_{im_ik})$. Analogously, we define the maximal nonrenewable resource requirement as $rmax^\nu_{ik} = \max_{m_i \in M_i}(r^\nu_{im_ik})$. A mode m_i of activity i is *non-executable* with respect to the nonrenewable resource type k when the following condition

Algorithm 2 Branching on mode and delaying alternatives

STEP 1 (**Next decision time**):
if $n \in \mathcal{A}_l$ store the current solution and go to Step 7
calculate $t_l = \min\left(\min_{i\in\mathcal{A}_l}\left(s'_i + d_{im'_i}\right), \min_{i\in N\setminus\mathcal{S}_l}(\tau_i)\right)$
set $\mathcal{A}_l = \mathcal{A}_l \setminus \{i \in \mathcal{A}_l \,|\, s'_i + d_{im'_i} = t_l\}$

STEP 2 (**Determine and start eligible activities**):
calculate $\mathcal{P}_l = \{j \in (N \setminus \mathcal{S}_l) \,|\, \forall (i,j) \in A : i \in \mathcal{S}_l$ and
$s'_i + d_{im'_i} \leq t_l\}$
calculate $\mathcal{E}_l = \{i \in \mathcal{P}_l \,|\, t_l \geq \tau_i\}$
if $\mathcal{E}_l = \emptyset$, go to Step 1, else start the activities in $\mathcal{E}_l$:
set $\mathcal{S}_l = \mathcal{S}_l \cup \mathcal{E}_l$, $\mathcal{A}_l = \mathcal{A}_l \cup \mathcal{E}_l$ and $\forall i \in \mathcal{E}_l$, set $s'_i = t_l$

STEP 3 (**Calculate mode alternatives**):
determine $\mathcal{E}'_l \subseteq \mathcal{E}_l$ as the set of eligible activities for which no mode has been assigned at a higher level $l' < l$ of the tree.
calculate $\mathcal{M}_l = \{\mathcal{M}_{l1}, \ldots, \mathcal{M}_{lp}\}$, the set of mode alternatives for $\mathcal{E}'_l$ that do not violate the nonrenewable resource constraints.

STEP 4 (**Select next mode alternative**):
if no untested mode alternative remains, go to Step 7
select the next untested mode alternative $\mathcal{M}_{lj} \in \mathcal{M}_l$
execute the mode alternative: $\forall i \in \mathcal{E}'_l$: set $m'_i = \mathcal{M}_{lj}(i)$
if $\exists k \in \mathcal{K}^\rho : \sum_{i\in\mathcal{A}_l} r^\rho_{im'_i k} > a^\rho_k$, go to Step 5
else set $\mathcal{S}_{l+1} = \mathcal{S}_l$, $\mathcal{A}_{l+1} = \mathcal{A}_l$, $l = l + 1$ and go to Step 1

STEP 5 (**Calculate minimal delaying alternatives**):
calculate $\mathcal{D}_l = \{\mathcal{D}_{l1}, \ldots, \mathcal{D}_{lq}\}$, the set of minimal delaying alternatives

STEP 6 (**Select next minimal delaying alternative**):
if no untested minimal delaying alternative remains, go to Step 4
select the next untested minimal delaying alternative $\mathcal{D}_{lj} \in \mathcal{D}_l$
set $\mathcal{S}_{l+1} = \mathcal{S}_l \setminus \mathcal{D}_{lj}$, $\mathcal{A}_{l+1} = \mathcal{A}_l \setminus \mathcal{D}_{lj}$, $l = l + 1$ and go to Step 1

STEP 7 (**Backtracking**):
set $l = l - 1$. If $l = 0$, STOP; else go to Step 6

Algorithm 3 Lower bound

Input: partial reactive schedule PS_l
Output: lower bound LB

$\forall i \in N \setminus \mathcal{S}_l$: determine the shortest mode $m_i^* \in M_i$ such that:
$r^{\nu}_{im_i^*k} + \sum_{j \in \mathcal{S}_l} r^{\nu}_{jm'_jk} \leq a^{\nu}_k, \quad \forall k \in \mathcal{K}^{\nu}$
if no such mode exists, set $LB = \infty$ and STOP
Given the partial reactive schedule PS_l, determine the earliest starting times $es_i \geq \tau_i$, $\forall i \in N \setminus \mathcal{S}_l$, assuming that these activities are executed in their shortest feasible modes m_i^* and ignoring the renewable resource constraints.
$LB = \sum_{i \in \mathcal{S}_l} \left[w_i(s'_i - s_i) + c_{im'_i} \right] + \sum_{i \in N \setminus \mathcal{S}_l} w_i(es_i - s_i)$

holds:

$$r^{\nu}_{im_ik} + \sum_{i \in N \setminus \{i\}} rmin^{\nu}_{ik} > a^{\nu}_k \tag{2}$$

A mode m_i of activity i is called *inefficient* if there exists an other mode $\tilde{m}_i \neq m_i$ such that $d_{i\tilde{m}_i} \leq d_{im_i}$, $r^{\rho}_{i\tilde{m}_ik} \leq r^{\rho}_{im_ik}$ for every renewable resource type $k \in \mathcal{K}^{\rho}$ and $r^{\nu}_{i\tilde{m}_ik} \leq r^{\nu}_{im_ik}$ for every nonrenewable resource type $k \in \mathcal{K}^{\nu}$. A nonrenewable resource type k is called *redundant* if $\sum_{i \in N} rmax^{\nu}_{ik} \leq a^{\nu}_k$. A structured approach for removing redundant nonrenewable resource types and inefficient or non-executable modes can be found in Sprecher et al. (1997).

When the preprocessing steps mentioned above have been performed, we can further process the input data by reducing the both the availabilities and the requirements of the nonrenewable resource types. More specifically, for an activity i we can decrease the nonrenewable resource requirement of every mode $m_i \in M_i$ as follows: $r^{\nu}_{im_ik} = r^{\nu}_{im_ik} - rmin^{\nu}_{ik}$. This decrease has to be reflected in the nonrenewable resource availability, so we set $a^{\nu}_k = a^{\nu}_k - rmin^{\nu}_{ik}$. We can do this for every activity i and for all nonrenewable resource types $k \in \mathcal{K}^{\nu}$. By doing so, nonrenewable resource conflicts will be detected earlier in the search process.

3.3.2 Left-shift dominance rule

Another well-known dominance rule that can be readily applied to our reactive scheduling problem is the left-shift rule (see e.g. Hartmann and Drexl (1998)). Indeed, because our objective function is a regular performance measure, left-shifting an activity can never lead to an increase in the rescheduling

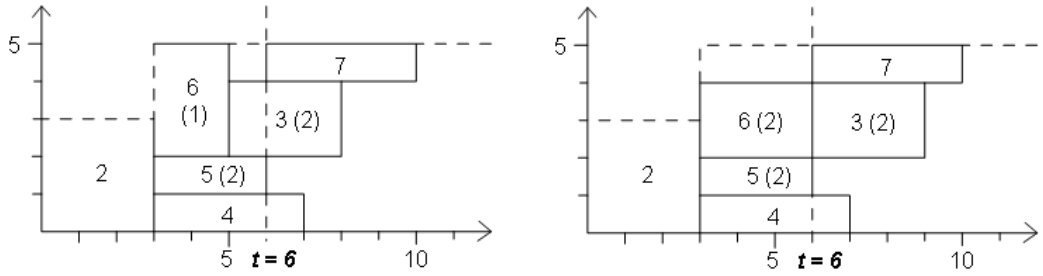
cost. The variant we use is known as the *single-mode local left-shift rule*. This rule states that a partial reactive schedule in which an activity i can be locally left-shifted without changing its mode and without violating the railroad scheduling constraint $s'_i \geq \tau_i$ and the renewable resource constraints, is dominated. This rule can be checked in Step 6 of Algorithm 2, before branching into the next untested minimal delaying alternative. For details on the implementation of such a left-shift rule, we refer the interested reader to Demeulemeester and Herroelen (1992).

3.3.3 Cutset rule

The so-called cutset rule is a very powerful dominance rule. It has been successfully applied in a wide variety of project scheduling problems (see e.g. Demeulemeester and Herroelen (1992), Hartmann and Drexl (1998) and Demeulemeester et al. (2000)). The gist of any cutset rule is that, given the partial reactive schedule obtained in the current node of the search tree, no better solutions will be found when branching from this node than the ones that have been obtained by branching from a certain previously visited node. Given a partial reactive schedule PS_l, we define a cutset as $\mathcal{S}_l$, the set of scheduled activities in PS_l. When denoting the starting time, finish time and mode of an activity $i \in PS_l$ as s'_{il}, f'_{il} and m'_{il}, respectively, we can formulate the following theorem:

Theorem 1 (Cutset dominance)**.** *A cutset $\mathcal{S}_b$ is dominated by a previously saved cutset $\mathcal{S}_a$ encountered on a different path of the search tree, if all of the following conditions are satisfied:*

- $\mathcal{S}_a = \mathcal{S}_b$ *and* $t_a \leq t_b$;
- $\displaystyle\sum_{i \in \mathcal{S}_a} r^{\nu}_{im'_{ia}k} \leq \sum_{j \in \mathcal{S}_b} r^{\nu}_{jm'_{jb}k}\,, \quad \forall k \in \mathcal{K}^{\nu}$;
- $\forall i \in \mathcal{A}_a \setminus \mathcal{A}_b : f'_{ia} \leq t_b$;
- $\forall i \in \mathcal{A}_a \cap \mathcal{A}_b : f'_{ia} \leq f'_{ib}$, *and* $m'_{ia} = m'_{ib}$;
- $$\sum_{i \in \mathcal{A}_a \setminus \mathcal{A}_b} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] + \sum_{i \in \mathcal{S}_a \setminus \mathcal{A}_a} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] \leq$$
$$\sum_{i \in \mathcal{A}_b \setminus \mathcal{A}_a} \left[w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right] + \sum_{i \in \mathcal{S}_b \setminus \mathcal{A}_b} \left[w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right]. \qquad (3)$$



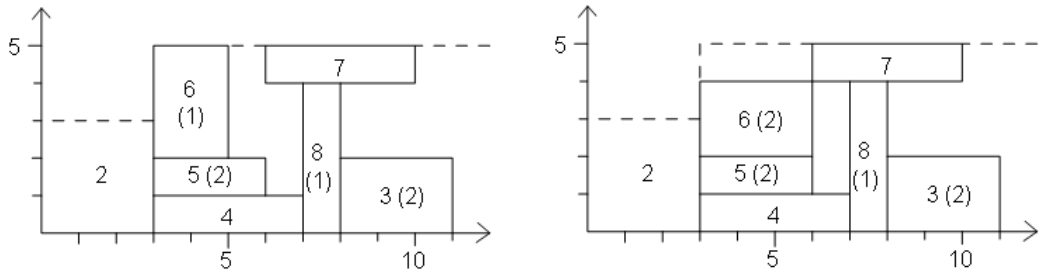
(a) Saved cutset $\mathcal{S}_a$. (b) Later encountered cutset $\mathcal{S}_b$.

Figure 4: Two example cutsets.

Proof: see Appendix 1. If the above conditions are satisfied, then any schedule that can be obtained by completing the partial reactive schedule PS_b can also be obtained by completing PS_a with no higher resulting rescheduling cost. Hence, no better solutions will be found by completing PS_b, and the node associated with the cutset $\mathcal{S}_b$ can be fathomed.

An example will illustrate the working logic of the cutset dominance rule. Two partial reactive schedules that are obtained during the reactive branch-and-bound procedure in response to a resource disruption are shown in Figure 4. For the network representation of the project and the project baseline schedule, refer again to Figure 1. At time instant $t_a = 6$ the partial reactive schedule PS_a shown in Figure 4 (a) is obtained, along with its associated cutset $\mathcal{S}_a = \{1, 2, 3, 4, 5, 6, 7\}$. Later during the search, the partial reactive schedule PS_b shown in Figure 4 (b) is observed. It is easy to see that this schedule is encountered on a different path than the partial reactive schedule of Figure 4 (a): the two schedules result from a different mode alternative for activity 6. We can now evaluate the conditions for cutset dominance:

- $\mathcal{S}_a = \mathcal{S}_b = \{1, 2, 3, 4, 5, 6, 7\}$, and the condition on the decision times t_a and t_b is satisfied: $6 \leq 6$;
- $\mathcal{K}^\nu = \emptyset$, so no conditions on nonrenewable resources need to be checked;
- $\mathcal{A}_a = \mathcal{A}_b = \{3, 4, 7\} \Rightarrow \mathcal{A}_a \setminus \mathcal{A}_b = \emptyset$;
- $\mathcal{A}_a \cap \mathcal{A}_b = \{3, 4, 7\}$:
  - $f'_{3a} \leq f'_{3b}$ and $m'_{3a} = m'_{3b}$, as required;
  - $f'_{4a} \leq f'_{4b}$ and $m'_{4a} = m'_{4b}$, as required;
  - $f'_{7a} \leq f'_{7b}$ and $m'_{7a} = m'_{7b}$, as required.



(a) Optimal continuation of PS_a. (b) Optimal continuation of PS_b.

Figure 5: Optimal continuations of PS_a and PS_b

- Finally, we check the condition on the rescheduling costs by calculating the left hand side of Eq. (3):

  $$\sum_{i \in \mathcal{A}_a \setminus \mathcal{A}_b} \left(w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right) + \sum_{i \in \mathcal{S}_a \setminus \mathcal{A}_a} \left(w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right) =$$

  $$\sum_{i \in \{2,5,6\}} \left(w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right) = c_{61} = 1.$$

  The right hand side evaluates to

  $$\sum_{i \in \mathcal{A}_b \setminus \mathcal{A}_a} \left(w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right) + \sum_{i \in \mathcal{S}_b \setminus \mathcal{A}_b} \left(w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right) =$$

  $$\sum_{i \in \{2,5,6\}} \left(w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right) = 0.$$

  We have $1 > 0$, so Equation (3) is not satisfied.

For this example, cutset dominance cannot be invoked. This might come as a surprise, since activity 3 finishes later in Figure 4 (b) than in Figure 4 (a) and the *total* rescheduling costs are equal for both partial reactive schedules. Indeed, the total rescheduling cost amounts to $5 * w_3 + c_{61} = 6$ for PS_a and $6 * w_3 = 6$ for PS_b.

Some further insight in the condition expressed by Equation (3) can be gained by looking at the optimal continuations of the partial reactive schedules PS_a and PS_b depicted in Figure 5. As can be seen in Figure 5, an optimal continuation of both partial reactive schedules will involve starting the highly inflexible activity 8 ($w_8 = 12$, see Table 1) as soon as possible, at the cost of further delaying the flexible activity 3 ($w_3 = 1$). Hence, the mode switch of activity 6 in Figure 5 (a) serves no purpose, and only introduces an additional cost. As a result, the partial reactive schedule PS_b could not be dominated by the partial reactive schedule PS_a.

The integration of the cutset dominance rule in Algorithm 2 can be achieved by modifying Step 2 as follows:

STEP 2 (**Determine and start eligible activities**):
calculate $\mathcal{P}_l = \{j \in (N \setminus \mathcal{S}_l) \,|\, \forall (i,j) \in A : i \in \mathcal{S}_l \text{ and } s'_i + d_{im'_i} \leq t_l\}$
calculate $\mathcal{E}_l = \{i \in \mathcal{P}_l \,|\, t_l \geq \tau_i\}$; if $\mathcal{E}_l = \emptyset$, go to Step 1
else calculate $\tilde{\mathcal{E}}_l \subseteq \mathcal{E}_l$: the set of eligible activities to which a mode has been assigned at some level $l' < l$ of the tree;
set $\mathcal{E}_l = \mathcal{E}_l \setminus \tilde{\mathcal{E}}_l$ and start the activities in $\tilde{\mathcal{E}}_l$: set $\mathcal{S}_l = \mathcal{S}_l \cup \tilde{\mathcal{E}}_l$, $\mathcal{A}_l = \mathcal{A}_l \cup \tilde{\mathcal{E}}_l$ and $\forall i \in \tilde{\mathcal{E}}_l$, set $s'_i = t_l$
if $\nexists k \in \mathcal{K}^\rho : \sum_{i \in \mathcal{A}_l} r^\rho_{im'_i k} > a^\rho_k$, check for cutset dominance.
If $\mathcal{S}_l$ is dominated, go to Step 7, else save the cutset $\mathcal{S}_l$, the decision time t_l, the set of active activities $\mathcal{A}_l$, their starting times s'_i and execution modes m'_i and the nonrenewable resource requirements req^ν_k of the partial reactive schedule PS_l, defined as $req^\nu_k = \sum_{i \in \mathcal{S}_l} r^\nu_{im'_i k}, \forall k \in \mathcal{K}^\nu$.
start the remaining eligible activities:
set $\mathcal{S}_l = \mathcal{S}_l \cup \mathcal{E}_l$, $\mathcal{A}_l = \mathcal{A}_l \cup \mathcal{E}_l$ and $\forall i \in \mathcal{E}_l$, set $s'_i = t_l$

As can be seen from the revised Step 2, we check for cutset dominance *after* we restart any delayed activities (these are the activities in the set $\tilde{\mathcal{E}}_l$) but *before* we start any activities that have become eligible for the first time on the path from the root node to the current node and as a consequence, have not yet been assigned a mode.

3.3.4 Resource infeasibility rule

Throughout the search, it can happen that the mode assignments that are made in a partial reactive schedule PS_l render a feasible mode assignment (w.r.t. the nonrenewable resource constraints) impossible for the set of unscheduled activities $N \setminus \mathcal{S}_l$. In that case, the node corresponding to the partial reactive schedule PS_l needs no further exploration. This kind of infeasibility can be detected in a relatively efficient way by means of a breadth-first search procedure.

Let $U = N \setminus \mathcal{S}_l$ be the set of unscheduled activities, and suppose $U = \{i_1, i_2, \ldots, i_p\}$. The breadth-first search procedure we propose attempts to find a feasible mode alternative for U within the given residual nonrenewable resource availability. Initially, we start with the activity set $\{i_1\}$ and we identify all feasible mode alternatives. These mode alternatives will define the nodes at the first level of the search tree. In every subsequent layer of the tree, we add an additional activity to the set and construct new feasible

mode alternatives for this activity set from the mode alternatives identified in the parent nodes. In other words, nodes at level l of the tree correspond to mode alternatives for the activity set $\{i_1, i_2, \ldots, i_l\}$. When all the nodes of a certain level of the tree have been calculated, we can invoke two pruning rules:

- **Pruning rule 1:** nodes that exceed the residual resource availability may be discarded.
- **Pruning rule 2:** if a node requires at least as many nonrenewable resources as another node on the same level of the tree, the former node may be discarded.

We can now proceed to a more formal description of the search procedure. For every $i_q \in U$, determine $\tilde{M}_{i_q}$, the set of *minimal constraining modes* for activity i_q: $\tilde{M}_{i_q} = \{m_j \in M_{i_q} \,|\, \nexists\, m_l \in M_{i_q}, l \neq j$ such that $\forall k \in \mathcal{K}^\nu : r^\nu_{im_l k} \leq r^\nu_{im_j k}\}$. During the breadth-first procedure, minimal constraining mode alternatives will be built up for the sets of activities $\{i_1\}$, $\{i_1, i_2\}$, $\ldots$, $\{i_1, i_2, \ldots, i_p\}$. With the different mode assignments made in these mode alternatives $\mathcal{M}_j$ correspond sets of nonrenewable resource requirements. To represent these resource requirements, we will use vector notation. By r^ν_{im} we denote the n^ν-vector $(r_{im1}, \ldots, r_{imn^\nu})$, assuming there are n^ν nonrenewable resource types. Resource requirements for a mode alternative $\mathcal{M}_j$ will be represented by the n^ν-vector $req_j = (req_{j1}, \ldots, req_{jn^\nu})$, where req_{jk} represents the resource requirement of the mode alternative $\mathcal{M}_j$ with respect to the k^{th} resource type. We will use the componentwise operators $+$ and $\leq$ on the n^ν-vectors defined above. To avoid ambiguity, we provide their formal definition:

- $req_j + req_l \quad \equiv \quad (req_{j1} + req_{l1},\, req_{j2} + req_{l2},\, \ldots,\, req_{jn^\nu} + req_{ln^\nu})$
- $req_j \leq req_l \quad \Leftrightarrow \quad req_{jk} \leq req_{lk},\ \forall k \in \mathcal{K}^\nu$

When the residual nonrenewable resource availability at the current decision point is represented by the n^ν-vector $avail$, the details of the breadth-first procedure can be described as shown in Algorithm 4. The procedure uses two queues, $QUEUE_1$ and $QUEUE_2$, containing resource requirement vectors. $QUEUE_1$ contains all vectors corresponding to a certain level of the tree. The subsequent level of the tree is then calculated by removing one element from $QUEUE_1$, calculating its children and adding those children to $QUEUE_2$. Once all children have been calculated and the dominated children have been removed, $QUEUE_1$ is set equal to $QUEUE_2$ and the algorithm proceeds to the next level of the tree.

Algorithm 4 Resource infeasibility check

calculate $\tilde{M}_{i_1}$
$\forall m_j \in \tilde{M}_{i_1}$: add $r^{\nu}_{i_1 m_j}$ to $QUEUE_1$
for $i = i_2, i_3, \ldots, i_p$:
    $QUEUE_2 \leftarrow \emptyset$
    calculate $\tilde{M}_i$
    while $QUEUE_1$ not empty:
        identify req, the first element in $QUEUE_1$
        remove req from $QUEUE_1$
        $\forall m_j \in \tilde{M}_i$:
            calculate $req_j = req + r^{\nu}_{im_j}$
            if $req_j \leq avail$, add req_j to $QUEUE_2$ (pruning rule 1)
    $\forall\, req_j,\, req_l \in QUEUE_2$:
        if $req_j \leq req_l$, remove req_l from $QUEUE_2$ (pruning rule 2)
    $QUEUE_1 \leftarrow QUEUE_2$
if $QUEUE_1$ is empty, no feasible mode assignment exists

An example will clarify things. Suppose, for the sake of example, that we extend the example project of Figure 1 with two nonrenewable resource types. Furthermore, suppose that at a certain decision point we have $U = \{5, 6, 7, 8, 9\}$. Note that, because of the data reduction dominance rules described in Section 3.3.1, activities with only a single mode always have zero consumption of nonrenewable resources after data reduction has been completed. Indeed, if only a single mode is present, the nonrenewable resource requirements for that mode become "sunk costs" and can be immediately subtracted from the total nonrenewable resource availability. With this in mind, we can reduce U to the following set of activities to be considered for the resource infeasibility rule: $U = \{5, 6, 8\}$. The nonrenewable resource requirements for these activities are given in Table 2.

Assume that the residual availabilities for the two nonrenewable resource types amount to $avail = (5, 5)$. Initially, the resource requirements of the minimal constraining modes corresponding to activity 5 are added to $QUEUE_1$. We find $\tilde{M}_5 = \{1, 2\}$, so we have $QUEUE_1 = \{(0, 2), (3, 1)\}$. These are the children of the root node in the breadth-first search tree, depicted in Figure 6. Every pass through the **for**-loop in Algorithm 4 now corresponds to a subsequent layer in the search tree.

During the first pass through the **for**-loop, activity 6 is considered. For activity 6, we find $\tilde{M}_6 = \{1, 2\}$. Both elements of $QUEUE_1$ are combined with the resource requirements of the two minimal constraining modes of activity 6. Element $(0, 2)$ is considered first. Adding the resource requirements of the

	activity 5		activity 6		activity 8	
m	r^{ν}_{5m1}	r^{ν}_{5m2}	r^{ν}_{6m1}	r^{ν}_{6m2}	r^{ν}_{8m1}	r^{ν}_{8m2}
1	0	2	3	0	2	3
2	3	1	0	1	2	5

Table 2: Nonrenewable resource requirements

two minimal constraining modes of activity 6 yields two new elements: $(3, 2)$ and $(0, 3)$. Both requirements can be satisfied by the residual availability $(5, 5)$, so both elements are added to $QUEUE_2$. Now the second element of $QUEUE_1$ is considered. Adding its resource requirement to the resource requirement of the first minimal constraining mode of activity 6 yields the vector $(6, 1)$, which exceeds the residual availability of $(5, 5)$. As a result, element $(6, 1)$ is not added to $QUEUE_2$. Considering the second minimal constraining mode of activity 6 yields the vector $(3, 2)$. Checking pruning rule 1, we have $(3, 2) \leq (5, 5)$, so this vector can be added to $QUEUE_2$. $QUEUE_1$ is now empty, so we proceed by checking the second pruning rule. We find $(3, 2) \leq (3, 2)$, so one of those vectors can be removed (see again Figure 6). We now have $QUEUE_2 = \{(0, 3), (3, 2)\}$, $QUEUE_1$ is set equal to $QUEUE_2$, and we proceed to the final pass through the **for**-loop.

In the final pass through the **for**-loop, activity 8 is considered. We find $\tilde{M}_8 = \{1\}$, because $(2, 3) \leq (2, 5)$. Activity 8 has only one minimal constraining mode, so in total, we calculate two child nodes. We find $(0, 3)+(2, 3) \not\leq (5, 5)$, and $(3, 2)+(2, 3) \leq (5, 5)$. This last node is accepted, and we have a feasible mode alternative for the activities in U. In this mode alternative, activities 5 and 6 are executed in mode 2, while activity 8 is executed in mode 1. This mode alternative has a total nonrenewable resource requirement equal to $(5, 5)$.

The resource infeasibility rule can be checked in Step 4 of Algorithm 2, when selecting the next mode alternative.

3.4 Search strategy

3.4.1 Regular branch-and-bound

Using the branching scheme presented in Algorithm 2 and the dominance rules described above, we can construct a classical branch-and-bound procedure, selecting at each level l of the search tree the node with the lowest lower bound and branching from that node in a depth-first fashion. This approach usually works fine, but for this particular scheduling problem, other approaches might work better. There are a number of reasons for this:

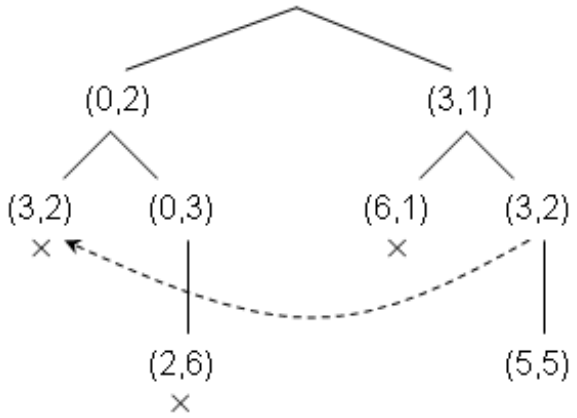


Figure 6: Breadth-first search tree

- The basic MRCPSP (and hence also the reactive scheduling problem) is computationally highly intractable. When n non-dummy activities can be scheduled in m different modes, this results in a total of m^n possible mode alternatives, each of which can be seen as an instance of the basic RCPSP.

- The lower bound we propose (see Algorithm 3) is very weak, because no strong assumptions can be made about the modes of the unscheduled activities. Moreover, Algorithm 2 does not select mode alternatives in an intelligent way. We cannot use the lower bound for selecting mode alternatives, because if we did, preference would always be given to modes yielding short activity durations, as renewable resource constraints are ignored in the lower bound.

- Due to the structure of the problem (i.e., the presence of activity weights and the nature of the objective function), large parts of the tree will contain only inferior solutions. Bad branching choices (either with respect to mode alternatives or with respect to delaying alternatives) can lead us into such a region. If this happens early in the search process, we will be wasting a lot of time finding nothing but bad solutions. Moreover, after finishing the exploration of such a region, the upper bound will be still so high that hardly any progress will have been made.

The above considerations, along with empirical results confirming these, have led us to explore other search strategies that try to avoid these problems.

3.4.2 Iterative Deepening A^*

Iterative Deepening A^* or *IDA** (Korf (1985)) is a tree-based optimal search technique that is related to best-first search. Given a root node *root*, a lower bound function LB and a function *generate_children* that generates the child nodes of a given node, the *IDA** procedure can be formulated as shown in Algorithm 5. The procedure *IDA** uses an auxiliary procedure *lb_limited_search*, shown in Algorithm 6.

Algorithm 5 *IDA**

```
lb ← LB(root)
while (no leaf reached)
    lb_new ← lb_limited_search(lb)
    lb ← lb_new
```

Algorithm 6 *lb_limited_search*

Input: lower bound *lb*
Output: either a new lower bound *lb_new* or an optimal leaf node

```
QUEUE ← {root}
lb_new ← ∞
while ((QUEUE ≠ ∅ and (no leaf reached))
    node ← first node in QUEUE
    remove node from QUEUE
    C ← generate_children(node)
    ∀c ∈ C :
        if c is a leaf: STOP with the optimal solution c
        else if LB(c) ≤ lb: add c to the front of QUEUE
            else lb_new = min(lb_new, LB(c))
```

Basically, the algorithm proceeds by *lb*-contours. Given a current lower bound *lb*, a depth-first branch-and-bound is initiated, discarding all nodes with a lower bound greater than *lb*. If no leaf node is found, the nodes that were discarded are used to determine a new lower bound $lb\_new > lb$. As no *lb*-contours are skipped, the first leaf node that is found during the search will contain an optimal solution to our problem. In that manner, *IDA** resembles best-first search, but avoids prohibitive memory use by branching in a truncated depth-first manner. The obvious drawback of the procedure is that large portions of the tree may be generated more than once.

The branching scheme presented in Algorithm 2 and the dominance rules described in Section 3.3 all remain valid when using *IDA** as a search strategy.

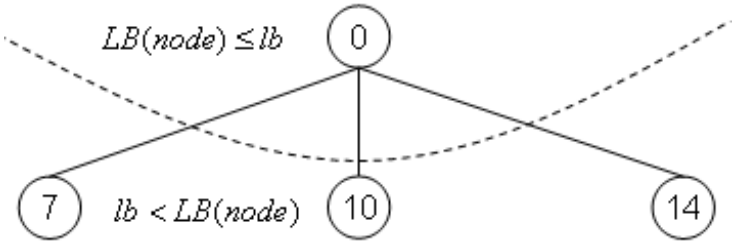


Figure 7: Iterative deepening A^*: first lb-contour

Note, however, that all cutset information must be discarded between two subsequent *lb_limited_search* passes. For a graphical illustration of the procedure, refer to Figure 7. A partial search tree is represented, with the lower bounds indicated inside the nodes. When the *IDA** procedure starts, the initial value for lb will be equal to $LB(root) = 0$. The children of the root node will be generated one by one and all of the children will be discarded because for every child c we have $LB(c) > lb$. The boundary between the accepted nodes and the rejected nodes is indicated by the dashed line. As no leaf node has been found among the accepted nodes, the first *lb_limited_search* pass will end with a new lower bound $lb\_new = \min(7, 10, 14) = 7$. This value is returned to the *IDA** procedure, and a new *lb_limited_search* pass will be initiated with this updated lower bound.

Note that no information regarding previous lb-contours is ever stored, so the second pass of *lb_limited_search* will proceed again from scratch, the only difference being a higher value for lb. The generation of the second lb-contour is illustrated in Figure 8. This time, the first child of the root node is accepted, and we branch from that node in a depth-first fashion. Its children will all be generated and rejected, and the procedure backtracks to the previous level in the tree. The two remaining child nodes of the root are generated and rejected, and the procedure finishes with $lb\_new = \min(14, 9, 13, 10, 14) = 9$. This value is returned to the *IDA** procedure, and a new *lb_limited_search* pass will be started with $lb = 9$. The *IDA** procedure stops when a leaf node has been found.

A number of conditions must be satisfied for *IDA** to be an appropriate search strategy for the problem at hand. An important condition is that the number of lb-contours be limited. Indeed, in the worst case, the value lb is incremented one by one, and the size of the tree generated in subsequent *lb_limited_search* passes only increases with a handful of nodes. To avoid this phenomenon, we have developed a technique we call *look-ahead check*. The idea of this technique is illustrated in Figures 9 and 10. A user defined parameter Δ must be supplied that reflects the *desired* increase of lb between

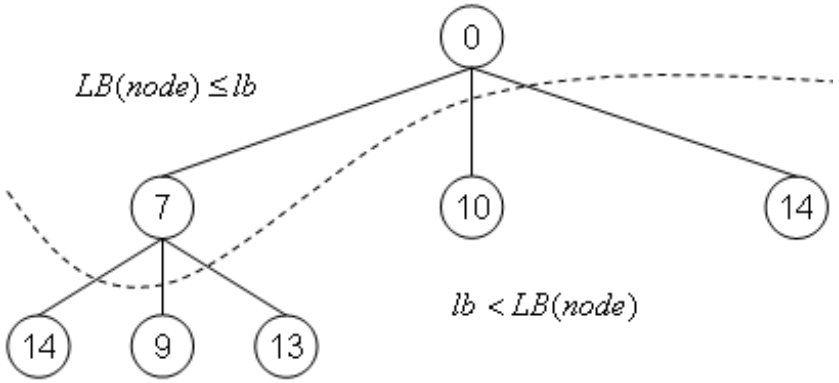


Figure 8: Iterative deepening A^*: second lb-contour

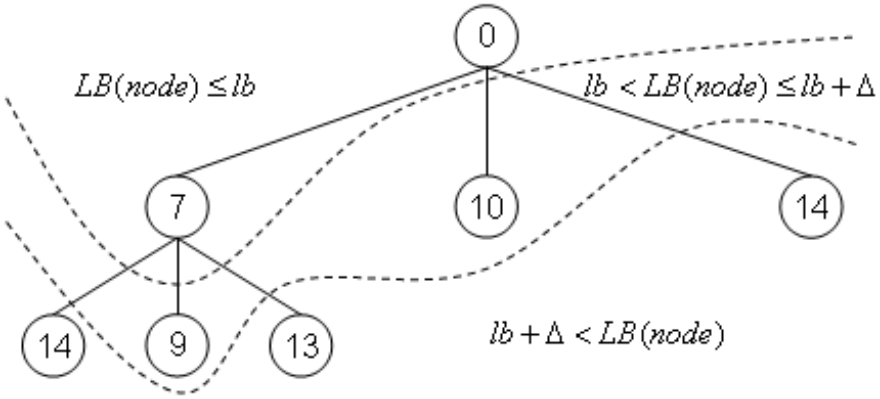


Figure 9: Look-ahead check: three lb-regions

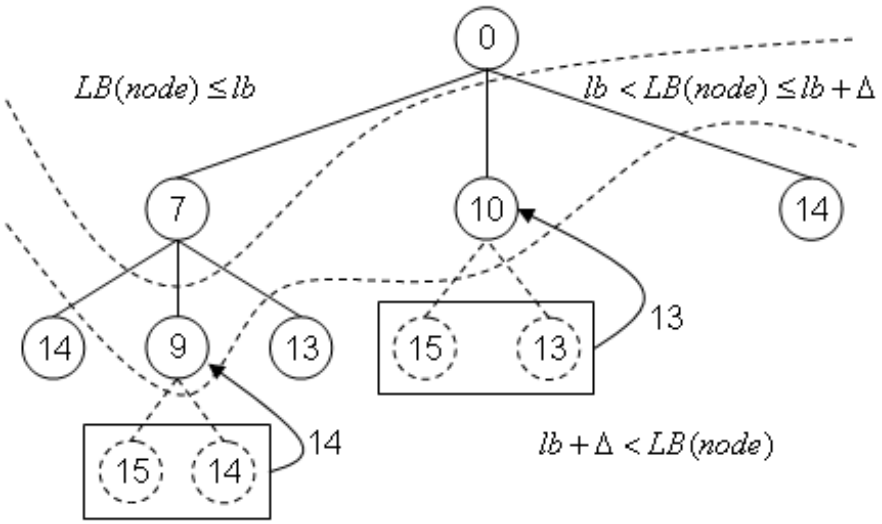


Figure 10: Look-ahead check: generating extra children

two subsequent passes of the procedure *lb_limited_search*. In Figure 9, it is assumed that $lb = 7$ and $\Delta = 4$. We can then divide the search tree into three regions. The upper region, as before, contains the nodes with a lower bound $LB(node) \leq lb$. These are the accepted nodes generated during a pass of the procedure *lb_limited_search*. The rejected nodes can be lumped into one of two categories. The first category contains the nodes with $lb < LB(node) \leq lb + \Delta$: this is the middle region shown in Figure 9, containing the nodes that would give rise to an undesirably small increase (given the parameter Δ) of lb. The remaining nodes belong to the second category: for these nodes, we have $lb + \Delta < LB(node)$. This is the lowermost region depicted in Figure 9.

The focus of the look-ahead check procedure will be on the nodes in the middle region. For these nodes, all the children will be generated up to a certain level. We can then calculate the minimum of the lower bounds of these children as an increased lower bound for the original node. The use of the look-ahead check is illustrated in Figure 10. Again we consider a pass of the procedure *lb_limited_search* with $lb = 7$. As was also the case in Figure 9, the nodes with lower bounds equal to respectively 14, 9, 13, 10 and 14 will be rejected. Before calculating the value of *lb_new* we will now identify those rejected nodes for which the condition $lb < LB(node) \leq lb + \Delta$ holds. Two nodes satisfy this condition, namely the nodes with $LB(node)$ equal to 9 and 10, respectively. Generating the children of the node with $LB(node) = 9$, are able to increase the lower bound value of that node to a value of $\min(15, 14) = 14$. For the node with $LB(node) = 10$, we find an

updated lower bound equal to $\min(15, 13) = 13$. When we now calculate the minimum lower bound value of the rejected nodes using the updated lower bound values, we find $lb\_new = \min(14, 14, 13, 13, 14) = 13$. By returning this increased value of lb_new, we are able to skip the contours corresponding to $lb = 9$ and $lb = 10$. Conceptually, the use of the look-ahead check will selectively increase the information value of the lower bound, leading to less contours encountered during *IDA** , and a larger number of nodes present within each contour (i.e., the contours are "thicker").

3.4.3 Branch-and-bound with tabu search

As will be shown in Section 4, both regular branch-and-bound and *IDA** have trouble solving certain harder instances of the reactive scheduling problem. Regular branch-and-bound may take very long before finding a first good solution (let alone an optimal one), while *IDA** sometimes grinds its way through a vast number of *lb*-contours (even with the look-ahead check refinement), thereby losing its possible advantage over regular branch-and-bound. In an attempt to combine the good parts of both branching strategies, we have developed a tabu search procedure (Glover (1989, 1990)) that searches for a heuristic solution for the reactive scheduling problem. The objective value of this heuristic solution can then be used as an upper bound for the regular branch-and-bound procedure. If the tabu search procedure finds a relatively good solution, we reduce the threat posed by large inferior regions, while eliminating the excessive revisiting of large portions of the tree entailed by the use of an *IDA** procedure.

The tabu search procedure we propose relies on fixing the modes of all activities that are unscheduled at the time of the disruption, and subsequently solving this reactive scheduling problem to optimality, using a "modeless" version of Algorithm 2. Indeed, if the modes are fixed, there is no need for branching on mode alternatives, so the solution space will be a lot smaller. Furthermore, the lower bound becomes stronger (since we know the modes of all unscheduled activities), and a number of conditions for cutset dominance (see Theorem 1) can be omitted or relaxed. All of this makes this problem considerably easier to solve in comparison to its multi-mode variant. As for the choice of branching strategy for this easier reactive scheduling problem, empirical results w.r.t. computation times have shown that regular branch-and-bound has the edge on *IDA** .

The detailed steps of the tabu search procedure are given in Algorithm 7. The procedure uses the function $modeless\_b\&b(\mathcal{M}, ub)$ to calculate the optimal objective value of the reactive scheduling problem given a mode alternative $\mathcal{M}$ and an upper bound ub on the rescheduling cost $\mathcal{C}$. If the

procedure $modeless\_b\&b(\mathcal{M}, ub)$ finds an optimal solution with an objective value $\mathcal{C}^* < ub$, the value $\mathcal{C}^*$ is returned; else, the value ∞ is returned. Other variables used in the tabu search procedure are the current iteration number $iter$, the maximum number of iteration steps max_iter and a list $\mathcal{L}$ indicating for every activity i an iteration number $\mathcal{L}(i)$ that reflects the tabu status of that activity: activity i is tabu (i.e. its mode can not be switched) as long as $\mathcal{L}(i) > iter$. The number of iteration steps an activity stays tabu is given by the parameter $tabu\_time$. The tabu search procedure we propose uses a diversification scheme. During the procedure we keep track of the number of times the mode of every activity i has been switched and store this information in the variables frq_i. We use these values to calculate a frequency penalty $\mathcal{P}_f$.

In order to solve the highly constrained project instances with respect to nonrenewable resources, we allow mode switches to result in a solution that is infeasible with respect to the nonrenewable resource constraints. By doing so, the tabu search procedure is able to escape from resource feasible "islands" in the solution space, allowing it to reach more promising regions with respect to the rescheduling costs. To nevertheless guide the search towards resource feasible solutions, the objective function value is penalized for nonrenewable resource infeasibility. This penalty $\mathcal{P}_\nu$ increases with increasing nonrenewable resource infeasibility ratio i^ν, as defined in Equation 4.

$$i^\nu = \frac{\sum_{k=1}^{n^\nu} \max\left(0, \frac{\left(\sum_{i=1}^{n} r_{i\mathcal{M}(i)k}\right) - a_k^\nu}{a_k^\nu}\right)}{n^\nu} \tag{4}$$

Clearly, we have $i^\nu = 0$ if the mode alternative $\mathcal{M}$ is resource feasible and $i^\nu > 0$ otherwise. The higher the value of i^ν, the higher the nonrenewable resource deficit. In Algorithm 7, $\mathcal{P}_\nu$ is calculated as $\mathcal{P}_\nu = (\kappa_1.i^\nu)^{\kappa_2}$. The nonrenewable resource infeasibility ratio i^ν is first scaled using a scaling factor κ_1 and then raised to the power of κ_2, resulting in a penalty that increases exponentially with increasing i^ν. Appropriate values for κ_1 and κ_2 were determined empirically in a computational experiment.

As a first step in the tabu search procedure, an initial mode alternative $\mathcal{M}$ is generated. In the case of a nonrenewable resource disruption, a local search is performed on the vector of baseline modes $(m_1, \ldots, m_n)$ to find a nonrenewable resource feasible mode assignment. In the case of a duration disruption or a renewable resource disruption, we simply use the vector of baseline modes. The so obtained mode alternative $\mathcal{M}$ is then used as a starting point for the actual tabu search procedure. During each step of the tabu search procedure a number of random non-tabu activities are elected for a mode switch. The size of the neighborhood (i.e., the number of activities

Algorithm 7 Tabu search

set initial mode alternative $\mathcal{M}$:
- in the case of a nonrenewable resource disruption:
  $\mathcal{M} \leftarrow local\_search(m_1, m_2, \ldots, m_n)$
- else: $\mathcal{M} \leftarrow (m_1, m_2, \ldots, m_n)$

$global\_best \leftarrow modeless\_b\&b(\mathcal{M}, \infty)$
$iter = 0$
$\forall i \in N$: set $\mathcal{L}(i) = 0$, $frq_i = 0$
while $(iter \neq max\_iter)$
  $\mathcal{N} \leftarrow$ generate random neighborhood of $\frac{n}{4}$ activities such that
    $\forall i \in \mathcal{N} : \mathcal{L}(i) \leq iter$
  $neigh\_best \leftarrow \infty$
  for all $i \in \mathcal{N}$:
    for all $m \in M_i \setminus \{\mathcal{M}(i)\}$:
      $old\_mode \leftarrow \mathcal{M}(i)$
      set $\mathcal{M}(i) = m$
      given $\mathcal{M}$, calculate i^ν according to Equation 4
      $\mathcal{P}_\nu \leftarrow (\kappa_1 . i^\nu)^{\kappa_2}$
      $\mathcal{P}_f \leftarrow frq_i$
      $\mathcal{C} \leftarrow modeless\_b\&b(\mathcal{M},\ neigh\_best - \mathcal{P}_\nu)$
      if $(i^\nu = 0)$ **and** $(\mathcal{C} < global\_best)$
        set $m^* = m$, $i^* = i$, $global\_best = \mathcal{C}$, $neigh\_best = \mathcal{C}$
      else
        calculate $\mathcal{C}' = \mathcal{C} + \mathcal{P}_\nu + \mathcal{P}_f$
        if $(\mathcal{C}' < neigh\_best)$
          set $m^* = m$, $i^* = i$, $neigh\_best = \mathcal{C}'$
        end if
      end if
      set $\mathcal{M}(i) = old\_mode$
    end for
  end for
  set $iter = iter + 1$
  set $\mathcal{M}(i^*) = m^*$, $\mathcal{L}(i^*) = iter + tabu\_time$, $frq_{i^*} = frq_{i^*} + 1$
end while

to be considered for a mode switch) has a large impact on the performance of the tabu search procedure. If we choose the neighborhood too small, we may fail to identify the more promising moves. If the neighborhood is too large, we may spend too much time per move, thereby limiting the number of moves made throughout a single run of the procedure. Empirical results have shown that a size of $\frac{n}{4}$ activities provides the best overall results.

Given the set of activities in the neighborhood, we determine the best mode switch taking into account the objective value, the penalties and the value of the best solution found so far. For every potential mode switch we solve the corresponding reactive scheduling problem to optimality using the adapted version of our regular branch-and-bound procedure. To speed up this branch-and-bound, we use an upper bound equal to the best objective value found so far in the current neighborhood, further corrected by subtracting the resource infeasibility penalty $\mathcal{P}_\nu$. The tabu search procedure terminates when the maximum number of iterations has been reached. Given the obtained heuristic objective value *global_best*, we can now start the regular branch-and-bound procedure described in Section 3.4.1, using *global_best* as upper bound.

3.4.4 Binary search

A fourth optimal reactive scheduling procedure combines all ideas of the previous sections. Given an upper bound ub and a lower bound lb on the rescheduling cost of a given problem instance, we can initiate a binary search within the interval $[lb, ub]$ to find the optimal solution. The structure of the binary search procedure is shown in Figure 11. First, the *IDA** procedure is executed to generate a lower bound. This procedure is stopped as soon as the generation of a contour exceeds a predefined time limit t_c. This results in *IDA** being allowed to calculate an arbitrarily strong lower bound, as long as the tightening of this bound can be performed within a reasonable computation time. If this results in *IDA** finding an optimal solution before the time-based stopping criterion is reached, we are done. If not, we have obtained a lower bound lb and proceed to the next step of the binary search procedure.

If a lower bound lb has been calculated without *IDA** finding an optimal solution, we will initiate a tabu search procedure (Algorithm 7) to generate an upper bound ub. What follows then is an iterative procedure that continues as long as $lb \neq ub$. Given lb and ub we calculate the midpoint $m = \lfloor \frac{lb+ub}{2} \rfloor$ and start the procedure $b\&b\_firstleaf$ with m as an upper bound on the rescheduling cost. This auxiliary procedure is the regular depth-first branch-and-bound version of Algorithm 2 with the important distinction that it stops

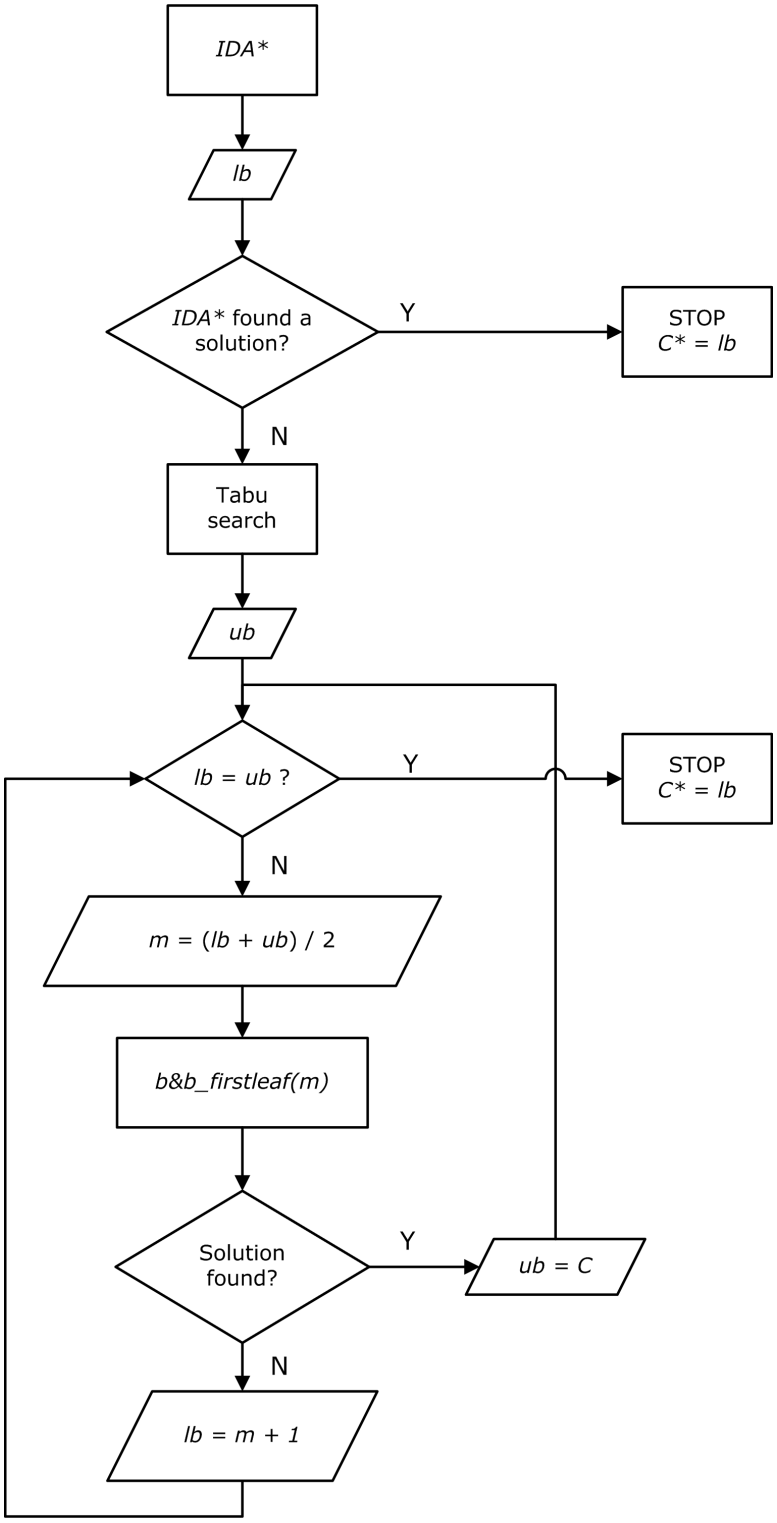


Figure 11: Binary search

when a first leaf node (and hence a solution with $\mathcal{C} \leq m$) has been found. The execution of this procedure can have one of two outcomes. Either a solution has been found, and we can update the upper bound as $ub = \mathcal{C}$, or no solution has been found, and we can set $lb = m + 1$. The iterative part of the binary search procedure stops when $lb = ub$ and we have an optimal solution with $\mathcal{C} = lb = ub$.

The binary search procedure combines some of the advantages of the other procedures. For easy instances (i.e. disrupted schedules that are "easy" to repair), *IDA** will in most cases outperform the other algorithms. By running *IDA** for a limited amount of time, we keep this important advantage. Also, the tabu search procedure combined with branch-and-bound may run into problems if the solution found by the tabu search procedure is rather bad when compared to the optimum. For these instances, the binary search will also perform well thanks to the "trial" upper bounds m that limit the search of the branch-and-bound procedure. Binary search will then behave like *IDA**, but the number of contours that need to be explored will never exceed $\log_2(ub - lb)$, whereas *IDA** may need to generate $\mathcal{C}^*$ contours in the worst case scenario.

4 Computational results

4.1 Computational set-up

To evaluate our procedures, we have generated baseline schedules for the 20-job and the 30-job multi-mode instances of the PSPLIB project scheduling library (Kolisch and Sprecher (1997)) using a dedicated tabu search procedure we developed for the basic MRCPSP. We did not generate optimal schedules, but schedules with a certain optimality gap when compared to the best known makespan reported in the literature. For the 554 20-job instances, we have generated baseline schedules with a makespan no higher than 30%, 20% and 10% above the optimal makespan. For the 552 30-job instances, no optimal makespans are known for some of the instances. Therefore, we have generated baseline schedules with a makespan no higher than 40%, 30% and 20% above the best known makespan reported in the literature. It should be clear that a larger optimality gap allows for more rescheduling flexibility, and hence results in an easier instance.

For a given instance set and baseline optimality gap, the procedures were tested by generating a single disruption scenario per instance as follows. The type of disruption generated is either a resource disruption or a duration disruption, both with an equal probability. In the case of a duration disrup-

tion, a random non-dummy activity i is selected from amongst the activities active during the first time period in the baseline schedule, and a duration increase Δ_i is generated as a uniformly distributed random variable drawn from the interval $[1, d_i]$. In the case of a resource disruption we first select whether the disruption affects a renewable or a nonrenewable resource type, both cases having an equal probability. For renewable resource disruptions, a random disruption is generated at time instant zero in such a way that the set of activities active during the first time period can no longer be executed as planned. The duration δ_t of the resource disruption is generated as a uniformly distributed random variable drawn from the interval $[1, 5]$. In the case of a nonrenewable resource disruption, we select a random nonrenewable resource type $k \in \mathcal{K}^{\nu}$ and we calculate the total nonrenewable resource requirement req_k of the baseline schedule. We then assume that the availability a_k^{ν} drops at time instant $t = 0$ to a value of $0.95 * req_k$, such that the baseline schedule is no longer resource feasible. Note that we deliberately choose for disruptions occurring during the first time periods of the execution of the project, as this will potentially result in the need to revise the entire schedule and hence in a harder reactive scheduling problem.

The inflexibility weights w_j for each non-dummy activity $j \in \{2, 3, ..., n-1\}$ are drawn from a discrete triangular distribution with $P(w_j = q) = (14 - \frac{11}{12}q)\%$ for $q \in \{1, 2....15\}$. This distribution results in a higher occurrence probability for low weights and in an average weight $w_{avg} = 5.4$. The weight w_n of the dummy end activity denotes the marginal cost of violating the project due date and is fixed at $\lfloor 10 \times w_{avg} \rfloor = 54$. For an extensive evaluation of the impact of the activity weights, we refer to Van de Vonder et al. (2005, 2006). The mode switching costs c_{im}, $\forall i \in N, m \in M_i \setminus \{m_i\}$ are generated as uniformly distributed random variables drawn from the interval $[0, 5]$. Note how we explicitly choose for mode switching costs that tend to be lower than the activity weights. In practice, meeting intermediate milestones and respecting the project deadline will always have very high importance. In that regard it will be advantageous to take mode-related measures (i.e., shifting resources) if this helps us to respect these milestones. Hence the motivation for the rather low mode switching costs in comparison to the activity weights.

All computational results have been obtained on a personal computer equipped with an Intel® Xeon® 2.33 GHz processor and the procedures have all been coded in C++.

Included refinements and dominance rules	♯ nodes	CPU time (s)	♯ contours
-	3681783	17.38	20
lac	1345085	5.69	11
lac + cs	90964	0.77	11
lac + cs + ls	56204	0.60	11
lac + cs + ls + ri	38388	0.48	11

Table 3: Effect of the dominance rules on the performance of the *IDA** procedure

4.2 Effect of the dominance rules

In this section, we study the effect of the various refinements and dominance rules for the proposed branching scheme on factors such as the computation time, the number of nodes visited and the number of contours generated throughout the branching procedure. More specifically, we study the incremental effect of the cutset dominance rule, the left shift dominance rule, the resource infeasibility rule and the look-ahead check refinement on the *IDA** procedure. In order to keep the experiment computationally tractable, we have used the rather easy-to-solve 20-job instances with a makespan optimality gap of 20%. With respect to disruption scenarios, the computational set-up described in the previous section was used.

First, we ran the *IDA** procedure with no dominance rules at all (except for the data reduction rule) and without the look-ahead check refinement. Then, we added the different enhancements and dominance rules one at a time, in the order of decreasing impact on the computation time. The results shown in Table 3 are averages over the 554 instances. Note that the column with the heading "♯ nodes" shows the average number of nodes visited during the entire *IDA** procedure, including any nodes that may be expanded in the context of the look-ahead check procedure.

In the second row, the results are shown for the algorithm with only look-ahead check (lac). We see that, when compared to the procedure without look-ahead check, the average number of contours drops dramatically from 20 to a value of 11. This is immediately reflected in a lower average number of nodes visited, and consequently, a lower average computation time. The next added dominance rule is the cutset dominance (cs) rule. Again we observe a very large decrease in the average number of nodes visited as well as the average computation time. The third and fourth most influential rules appear to be the left-shift dominance (ls) and the resource infeasibility (ri) rule. Both rules yield a smaller yet still important speed-up.

Baseline optimality gap	Regular B&B	*IDA**	B&B with tabu search	Binary search
30 %	0.44	0.14	0.38	0.10
20 %	0.54	0.48	0.52	0.28
10 %	0.81	1.30	0.77	0.80

Table 4: CPU times (s) for 20-job instances

4.3 Effect of the search strategy

In this section, we compare the performance of the four proposed algorithms (regular branch-and-bound, *IDA**, branch-and-bound with tabu search and binary search) with respect to computation times. More specifically, we compare the performance of these algorithms on 20- and 30-job instances, with a varying baseline optimality gap. Concerning the generation of inflexibility weights, rescheduling costs and disruptions, we have used the same experimental setup as in the previous section. For the 20-job PSPLIB instances, we have generated baseline schedules with makespans no more than 10%, 20% and 30% above the optimal makespan reported in the literature. The average computation times of the reactive procedures over these instances are shown in Table 4.

When looking at the results, a first thing to note is that for every procedure, the computation time increases with decreasing baseline optimality gap. This is no surprise, because near optimal baseline schedules will be harder to get back on track when a disruption occurs. In this computational experiment, the computation times of the different procedures are rather close to one another, but the binary search procedure appears to be the overall best performer. The procedure has the shortest computation times on the 30% and 20% optimality gap instances, and the second to shortest computation time on the hard 10% optimality gap instances.

We have also tested our procedures on the 552 30-job instances of PSPLIB. For these instances, we have generated baseline schedules with a makespan no more than 20%, 30% and 40% above the smallest obtained makespan reported in the literature. The average computation times over the 552 instances are shown in Table 5. On this instance set, the computation times of regular branch-and-bound increase very fast, and for the hard 20% instances, no solution was found within a reasonable time limit. *IDA** on the other hand performs surprisingly well and outperforms branch-and-bound with tabu search on the hardest instance set. But as is very clear in this experiment, binary search is the procedure of choice, yielding reasonable computation times even on the hardest instance sets.

Baseline optimality gap	Regular B&B	*IDA**	B&B with tabu search	Binary search
40 %	83.0	32.7	13.7	15.1
30 %	561.3	54.3	25.7	24.5
20 %	-	138.5	324.4	63.5

Table 5: CPU times (s) for 30-job instances

As a final note, it is worth to mention that we have also tested the procedures on 30-job instances with a 10% optimality gap, but for this set of instances, none of the procedures were able to finish within an acceptable time limit.

4.4 Effectiveness of the tabu search procedure

In this section, we study the performance of the tabu search procedure described in Section 3.4.3 as a stand-alone heuristic for the reactive scheduling problem. For the 20- and 30-job instances we have used the same computational set-up as described above, the only difference being that all resource disruptions are now renewable resource disruptions. Doing so will allow us to investigate what happens to the optimal objective values if we do not allow mode switches to be performed. For all instances and makespan optimality gaps, we have compared the objective values obtained by the tabu search procedure with the optimal objective values. The results are shown in Tables 6 (a) and (b). For every baseline optimality gap, we show the average rescheduling costs calculated by the exact procedure ($\mathcal{C}^*$), the average rescheduling costs obtained by the tabu search procedure ($\mathcal{C}_{TS}$), the average optimal rescheduling costs obtained when the reactive modes are fixed to the baseline modes ($\mathcal{C}^*_{fix}$) and the percentage of problem instances solved to optimality by the tabu search procedure.

The column $\mathcal{C}^*_{fix}$ is included to put the interpretation of the columns $\mathcal{C}^*$ and $\mathcal{C}_{TS}$ in a broader perspective. From the $\mathcal{C}^*_{fix}$-values we can see that a large reduction in rescheduling costs can be obtained when choosing for an optimal or near-optimal reactive policy instead of some myopic heuristic. Furthermore, we notice a relatively small gap between the average optimal rescheduling costs and the average rescheduling costs obtained by the tabu search procedure. Also, the average computation time required by the tabu search procedure is quite reasonable (17.6 seconds on the hardest instance set), and the procedure succeeds in solving a large part of the instances to optimality.

(a) Results for the 20-job instances.

Baseline optimality gap	$\mathcal{C}^*$	$\mathcal{C}_{TS}$	$\mathcal{C}^*_{fix}$	% optimal
30 %	52	56	173	64 %
20 %	61	64	170	69 %
10 %	87	89	178	73 %

(b) Results for the 30-job instances.

Baseline optimality gap	$\mathcal{C}^*$	$\mathcal{C}_{TS}$	$\mathcal{C}^*_{fix}$	% optimal
40 %	45	49	208	64 %
30 %	52	56	204	64 %
20 %	67	70	206	68 %

Table 6: Performance of the tabu search procedure.

4.5 Characteristics of the optimal reactive schedules

In the previous section we have seen that the possibility of mode switches as a reactive scheduling instrument can lead to a dramatic reduction of the rescheduling costs (refer again to the values for $\mathcal{C}^*$ and $\mathcal{C}^*_{fix}$ in Tables 6 (a) and (b)). Therefore, it is interesting to investigate some properties of the optimal reactive schedules with respect to mode switches. More specifically, we are interested in the average number of mode switches performed in optimal reactive schedules and in the percentage of these mode switches that results in an activity duration increase, an activity duration decrease or an equal activity duration. For the 20- and 30-job instances and the different makespan optimality gaps, we have calculated the average number of mode switches performed in a reactive schedule in response to either a renewable resource disruption or an activity duration disruption. For all instance sets, this number amounts to 10 % of the non-dummy activities. In other words, we observe two mode switches (on average) for the 20-job instances and three mode switches for the 30-job instances, and this is the case for every makespan optimality gap studied in the experiment.

A mode switch can result in a duration decrease or increase of the involved activity. If a mode switch results in an equal activity duration, we speak of a *resource alternative*. As a side note, we remark that in the instance sets used, approximately 23% of the non-dummy activities have at least two modes with an equal activity duration and can thus be subjected to a resource alternative. Given the activities subjected to a mode switch, we can calculate the proportion of mode switches resulting in a duration increase, a duration

(a) Results for the 20-job instances.

Baseline optimality gap	% duration decreases	% duration increases	% resource alternatives
30 %	76.7 %	15.5 %	7.8 %
20 %	69.8 %	21.2 %	9.0 %
10 %	64.1 %	26.0 %	9.9 %

(b) Results for the 30-job instances.

Baseline optimality gap	% duration decreases	% duration increases	% resource alternatives
40 %	81.4 %	11.2 %	7.4 %
30 %	75.4 %	16.3 %	8.3 %
20 %	70.0 %	19.7 %	10.3 %

Table 7: Types of mode switches in the optimal reactive schedules.

decrease and a resource alternative. The results of this analysis are shown in Tables 7 (a) and (b). Not surprisingly, most mode switches result in a duration decrease of the affected activity. What is more remarkable, is the fact that still a fairly large portion of the mode switches results in either a duration increase or a resource alternative. The proportion of duration increases and resource alternatives among the mode switches increases with decreasing makespan optimality gap, which is logical because on average, more efficient schedules will include a larger proportion of activities scheduled in their shortest mode. As a result, less activity crashes are possible in a reactive schedule. The figures in Tables 7 (a) and (b) reveal that some creative reallocation of resources effectively allows us to repair disrupted schedules. Activities with some remaining slack will be allowed to take a little longer to complete, so that the resources freed by this action can be allocated to the more "problematic" activities in the disrupted schedule.

5 Conclusions

In this paper, we have formulated a reactive scheduling problem for the multi-mode RCPSP. Given an MRCPSP baseline schedule and a resource disruption or an activity duration disruption that occurs during the execution of the baseline schedule, we want to obtain a reactive schedule that minimizes the rescheduling costs, defined as the sum of the mode switching costs and the costs incurred due to activity starting time deviations. We have indicated that the literature on reactive scheduling procedures for the MRCPSP is very

scarce.

We have proposed a branching scheme based on mode and delaying alternatives for optimally solving the reactive scheduling problem. A number of dominance rules for this branching scheme have been developed, namely a single-mode left-shift rule, a resource infeasibility rule and a cutset dominance rule. In order to circumvent certain problems posed by a regular branch-and-bound procedure, we have proposed an alternative branching strategy for this reactive scheduling problem, namely *IDA**. We have suggested a *look-ahead check* refinement to further speed up the *IDA** procedure. To take advantage of the insights obtained from testing regular branch-and-bound and *IDA** as branching strategies for the reactive scheduling problem, we have attempted to combine the good parts of both schemes into a single procedure. Therefore, we have proposed the use of a tabu search procedure to obtain a heuristic solution for the reactive scheduling problem, and to use the objective value of this heuristic solution as an upper bound to be used in the regular branch-and-bound procedure. Doing so combines the advantage of *IDA** (not generating inferior nodes) with the efficiency of regular branch-and-bound (generating every node only once). In a further effort to combine the good parts of the previous three exact procedures, we have developed an algorithm that uses *IDA**, regular branch-and-bound and tabu search to obtain the optimal solution through a binary search in the interval $[lb, ub]$, with lb and ub a lower and an upper bound on the optimal solution value, respectively.

We have tested the effect of the dominance rules and the branching strategy on 20- and 30-job benchmark instances. The results have shown the effectiveness of the look-ahead check refinement and the other dominance rules if used in the *IDA** procedure. When comparing the computation times of regular branch-and-bound, *IDA**, tabu search with branch-and-bound and binary search on instances of varying size and difficulty, we found that the binary search procedure is the most robust, yielding relatively short computation times on all instance sets used in the experiment. The algorithm effectively combines the advantages of regular branch-and-bound, *IDA** and tabu search in a single exact procedure. As for the performance of the tabu search procedure as a stand-alone reactive scheduling heuristic, we found that the average rescheduling costs obtained by the tabu search procedure only slightly deviate from the average optimal rescheduling costs. Also, the required computation time of the tabu search procedure remains within reasonable limits, even for the hardest instances considered in our experiments.

When evaluating the use of mode switches for dealing with schedule disruptions, we found two important conclusions. First of all, the use of mode switches can lead to a drastic reduction of the rescheduling costs, even in

baseline schedules that have a makespan deviating as little as 10 % from the theoretically shortest possible makespan obtainable for the given project network. In this way, the presence of multiple execution modes can prove to be a valuable reactive scheduling instrument, even when little express safety is included in the baseline schedule (in the form of a suboptimal makespan). Secondly, we found that optimal schedules not only use mode switches resulting in shorter activity durations, but also mode switches resulting in an equal or longer activity duration. In this way, resources are shifted from less "critical" activities (sometimes resulting in an activity duration increase) to the more problematic areas in the disrupted schedule.

Appendix 1

Proof of Theorem 1.
We assume that two cutsets $\mathcal{S}_a$ and $\mathcal{S}_b$ are given with accompanying partial reactive schedules PS_a and PS_b for which the conditions for cutset dominance hold (see Section 3.3.3). Suppose an optimal schedule S^* of activity starting times s_i^* and activity execution modes m_i^* is obtained by branching from PS_b, with a rescheduling cost equal to $\mathcal{C}^*$. We prove that a schedule $\tilde{S}$ can be obtained by branching from PS_a with a rescheduling cost not greater than $\mathcal{C}^*$. The proof consists of two parts. First, we show how the schedule $\tilde{S}$ can be constructed from PS_a. Then, we prove that the rescheduling cost of $\tilde{S}$ is not greater than $\mathcal{C}^*$.

First, remark that all activities in $\mathcal{S}_a$ have been assigned a mode, and this mode will not be altered during the branching procedure. Because of the cutset dominance condition on the nonrenewable resources, any mode assignment to the set of unscheduled activities $i \in N \setminus \mathcal{S}_b$ that can be achieved under the residual availabilities of PS_b can also be achieved under the residual availabilities of PS_a.

Construct a schedule $\tilde{S}$ from PS_a as follows. We divide the set of activities N into two disjoint sets: the unscheduled activities $N \setminus \mathcal{S}_a$ and the scheduled activities $\mathcal{S}_a$. We first focus on the scheduled activities. Because the mode assignments made for this set of scheduled activities are final, we must have $\tilde{m_i} = m_{ia}, \forall i \in \mathcal{S}_a$. Concerning the starting times of these activities in $\tilde{S}$, consider $\mathcal{A}_b$, a subset of the set of scheduled activities. For these activities $i \in \mathcal{A}_b$, we will fix the starting times $\tilde{s_i}$ depending on their "status" in PS_a, as follows:

- **Case 1.** $i \in \mathcal{A}_a$: by the cutset dominance conditions, we know that $(m_i^* =)\ m'_{ib} = m'_{ia}$, $f'_{ia} \leq f'_{ib}$ and hence $s'_{ia} \leq s'_{ib} (\leq s_i^*)$. Set $\tilde{s_i} = s_i^*$. This implies right-shifting activity i in the partial reactive schedule

corresponding with PS_a by an amount $s_i^* - s'_{ia}$. Note that no renewable resource conflict can arise from right-shifting an active activity in a partial reactive schedule.

- **Case 2.** $i \in \mathcal{S}_a \setminus \mathcal{A}_a$: activity i has already finished in PS_a, so the starting time can no longer be changed by branching from PS_a. We have $\tilde{s}_i = s'_{ia}$.

Note that for all activities $i \in \mathcal{A}_b$, we either have $\tilde{f}_i = f_i^*$ (Case 1) or $\tilde{f}_i \leq t_a \leq t_b \leq f'_{ib} \leq f_i^*$ (Case 2). Having fixed the starting times of the activities $i \in \mathcal{A}_b$, we still need to determine the starting times of the remaining scheduled activities, defined by the set $\mathcal{S}_b \setminus \mathcal{A}_b$. By the cutset dominance conditions, we know that the activities $i \in \mathcal{S}_b \setminus \mathcal{A}_b$ have either already finished in PS_a, or are active in PS_a but have a finish time $f'_{ia} \leq t_b$. In any case, these activities in PS_a will not impede the start of any of the *unscheduled* activities $j \in N \setminus \mathcal{S}_b$ that are all set to start at starting times $s_j^* \geq t_b$ in the optimal schedule S^*. We set $\tilde{s}_i = s'_{ia}$, $\forall i \in \mathcal{S}_b \setminus \mathcal{A}_b$.

Finally, we need to determine the modes and starting times of the unscheduled activities. We simply copy the starting times and modes from the optimal schedule: set $\tilde{s}_i = s_i^*$ and $\tilde{m}_i = m_i^*$, $\forall i \in N \setminus \mathcal{S}_b$. No precedence relations are violated by these decisions, because all predecessors of the unscheduled activities either finish not later than time instant t_b in PS_a, or they finish at the same time as the corresponding activity in S^*. Furthermore, no renewable resource conflict will arise by these decisions, because the optimal schedule is resource feasible, and because the set of activities active at time instant t_b in PS_a is a subset of $\mathcal{A}_b$, and the activities in this set do not finish later than the corresponding activities in $\mathcal{A}_b$.

In summary, to construct the schedule $\tilde{S}$ from PS_a, we keep the modes of PS_a for the scheduled activities and schedule the unscheduled activities in their "optimal" modes m_i^*. The starting times of the scheduled and finished activities in PS_a are left unchanged, the starting times of the activities in $\mathcal{A}_a \cap \mathcal{A}_b$ are set to the starting times in the optimal schedule, and the starting times of the activities in $\mathcal{A}_a \setminus \mathcal{A}_b$ are left unchanged. The starting times of the unscheduled activities are then copied from the optimal schedule.

To complete the proof, we show that the rescheduling cost $\tilde{\mathcal{C}}$ of $\tilde{S}$ is not greater than the rescheduling cost $\mathcal{C}^*$ of S^*. Therefore, we divide the set N of activities into four disjoint subsets. For the schedule $\tilde{S}$, we define the partition as follows:

$$N = (N \setminus \mathcal{S}_a) \cup (\mathcal{A}_a \cap \mathcal{A}_b) \cup (\mathcal{A}_a \setminus \mathcal{A}_b) \cup (\mathcal{S}_a \setminus \mathcal{A}_a) \tag{5}$$

For the schedule S^*, the partition is analogous:

$$N = (N \setminus \mathcal{S}_b) \cup (\mathcal{A}_a \cap \mathcal{A}_b) \cup (\mathcal{A}_b \setminus \mathcal{A}_a) \cup (\mathcal{S}_b \setminus \mathcal{A}_b) \tag{6}$$

The rescheduling costs incurred by both schedules can thus be obtained as the sum of the rescheduling costs caused by the activities in the four disjoint sets. By construction of the schedule $\tilde{S}$, we observe that the rescheduling costs caused by the activities in the first two sets are equal for both schedules, so we can formulate the following equivalence:

$$\tilde{\mathcal{C}} \leq \mathcal{C}^* \Leftrightarrow \sum_{i \in \mathcal{A}_a \setminus \mathcal{A}_b} [w_i(\tilde{s}_i - s_i) + c_{i\tilde{m}_i}] + \sum_{i \in \mathcal{S}_a \setminus \mathcal{A}_a} [w_i(\tilde{s}_i - s_i) + c_{i\tilde{m}_i}] \leq$$

$$\sum_{i \in \mathcal{A}_b \setminus \mathcal{A}_a} \left[w_i(s_i^* - s_i) + c_{im_i^*}\right] + \sum_{i \in \mathcal{S}_b \setminus \mathcal{A}_b} \left[w_i(s_i^* - s_i) + c_{im_i^*}\right] \tag{7}$$

By construction of the schedules S^* and $\tilde{S}$, we can rewrite Equation (7) as follows:

$$\tilde{\mathcal{C}} \leq \mathcal{C}^* \Leftrightarrow \sum_{i \in \mathcal{A}_a \setminus \mathcal{A}_b} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] + \sum_{i \in \mathcal{S}_a \setminus \mathcal{A}_a} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] \leq$$

$$\sum_{i \in \mathcal{A}_b \setminus \mathcal{A}_a} \left[w_i(s_i^* - s_i) + c_{im'_{ib}}\right] + \sum_{i \in \mathcal{S}_b \setminus \mathcal{A}_b} \left[w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right] \tag{8}$$

In Equation (8), the values of s_i^* will be unknown at the time the cutset information is calculated. We do know, however, that $s_i^* \geq s'_{ib}$, $\forall i \in \mathcal{A}_b \setminus \mathcal{A}_a$. Hence we can formulate the following sufficient condition on the rescheduling costs for cutset dominance to apply:

$$\sum_{i \in \mathcal{A}_a \setminus \mathcal{A}_b} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] + \sum_{i \in \mathcal{S}_a \setminus \mathcal{A}_a} \left[w_i(s'_{ia} - s_i) + c_{im'_{ia}}\right] \leq$$

$$\sum_{i \in \mathcal{A}_b \setminus \mathcal{A}_a} \left[w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right] + \sum_{i \in \mathcal{S}_b \setminus \mathcal{A}_b} \left[w_i(s'_{ib} - s_i) + c_{im'_{ib}}\right] \Rightarrow \tilde{\mathcal{C}} \leq \mathcal{C}^* \tag{9}$$

□

References

Alcaraz, J., Maroto, C. and Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, 54(6), pp 614–626.

Aytug, H., Lawley, M., McKay, K., Mohan, S. and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1), pp 86–110.

Buddhakulsomsiri, J. and Kim, D. (2006). Properties of multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research*, 175(1), pp 279–295.

Davenport, A. and Beck, J. (2002). A survey of techniques for scheduling with uncertainty. *Unpublished manuscript*.

De Reyck, B. and Herroelen, W. (1999). The multi-mode resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2), pp 538–556.

Demeulemeester, E., De Reyck, B. and Herroelen, W. (2000). The discrete time/resource trade-off problem in project networks: A branch-and-bound approach. *IIE Transactions*, 32, pp 1059–1069.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38, pp 1803–1818.

Drexl, A. and Grünewald, J. (1993). Non-preemptive multi-mode resource-constrained project scheduling. *IIE Transactions*, 25(5), pp 74–81.

Glover, F. (1989). Tabu search, Part I. *INFORMS, Journal of Computing*, 1, pp 190–206.

Glover, F. (1990). Tabu search, Part II. *INFORMS, Journal of Computing*, 2, pp 4–32.

Hartmann, S. (2001). Project scheduling with multiple modes: A genetic algorithm. *Annals of Operations Research*, 102, pp 111–135.

Hartmann, S. and Drexl, A. (1998). Project scheduling with multiple modes: A comparison of exact algorithms. *Networks*, 32, pp 283–297.

Hartmann, S. and Sprecher, A. (1996). A note on "Hierarchical models for multi-project planning and scheduling". *European Journal of Operational Research*, 94(2), pp 377–383.

Heilmann, R. (2003). A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 127(2), pp 348–365.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al.. *European Journal of Operational Research*, 128(3), pp 221–230.

Herroelen, W. and Leus, R. (2004). Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research*, 42(8), pp 1599–1620.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty – Survey and research potentials. *European Journal of Operational Research*, 165, pp 289–306.

Hildum, D. W. (1995). *Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems*. PhD thesis. Amherst, MA, USA.

Jozefowska, J., Mika, M., Rozycki, R., Waligora, G. and Weglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, 102, pp 137–155.

Kolisch, R. and Sprecher, A. (1997). PSPLIB – A project scheduling library. *European Journal of Operational Research*, 96, pp 205–216.

Korf, R. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1), pp 97–109.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2007). A tabu search procedure for developing robust predictive project schedules. *International Journal of Production Economics*, 111(2), pp 496–508.

Lambrechts, O., Demeulemeester, E. and Herroelen, W. (2008). Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of Scheduling*, 11(2), pp 121–136.

Lova, A., Tormos, P. and Barber, F. (2006). Multi-mode resource constrained project scheduling: Scheduling schemes, priority rules and mode selection rules. *Inteligencia artificial*, 30, pp 69–86.

Lova, A., Tormos, P., Cervantes, M. and Barber, F. (2008). A hybrid genetic algorithm for the multi-mode resource constrained project scheduling problem. *in* F. S. Serifoglu and U. Bilge (eds), *Proceedings of the Eleventh International Workshop on Project Management and Scheduling*. pp 189–192.

Sabzehparvar, M. and Seyed-Hosseini, S. M. (2008). A mathematical model for the multi-mode resource-constrained project scheduling problem with mode dependent time lags. *Journal of Supercomputing*, 44(3), pp 257–273.

Speranza, M. and Vercellis, C. (1993). Hierarchical models for multi-project planning and scheduling. *European Journal of Operational Research*, 64(2), pp 312–325.

Sprecher, A. and Drexl, A. (1998). Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *European Journal of Operational Research*, 107(2), pp 431–450.

Sprecher, A., Hartmann, S. and Drexl, A. (1997). An exact algorithm for project scheduling with multiple modes. *OR Spektrum*, 19, pp 195–203.

Van de Vonder, S. (2006). *Proactive-reactive procedures for robust project scheduling*. PhD thesis. Department of Decision Sciences and Information Management (KBI), K.U. Leuven.

Van de Vonder, S., Ballestín, F., Demeulemeester, E. and Herroelen, W. (2007). Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, 52(1), pp 11–28.

Van de Vonder, S., Demeulemeester, E. and Herroelen, W. (2008). Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3), pp 723–733.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. and Leus, R. (2005). The use of buffers in project management: The trade-off between stability and makespan. *International Journal of Production Economics*, 97, pp 227–240.

Van de Vonder, S., Demeulemeester, E., Herroelen, W. and Leus, R. (2006). The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2), pp 215–236.

Vieira, G., Herrmann, J. and Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1), pp 35–58.

Zhang, H., Tam, C. and Li, H. (2006). Multimode project scheduling based on particle swarm optimization. *Computer-Aided Civil and Infrastructure Engineering*, 21, pp 93–103.

Zhu, G., Bard, J. and Yu, G. (2005). Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56, pp 365–381.

Zhu, G., Bard, J. and Yu, G. (2006). A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing*, 18(3), pp 377–390.